

INTRODUCCION A LAS HERRAMIENTAS EN C# -

MESSAGEBOX Y EL DIALOG RESULT

Guía de laboratorio – Capítulo IV

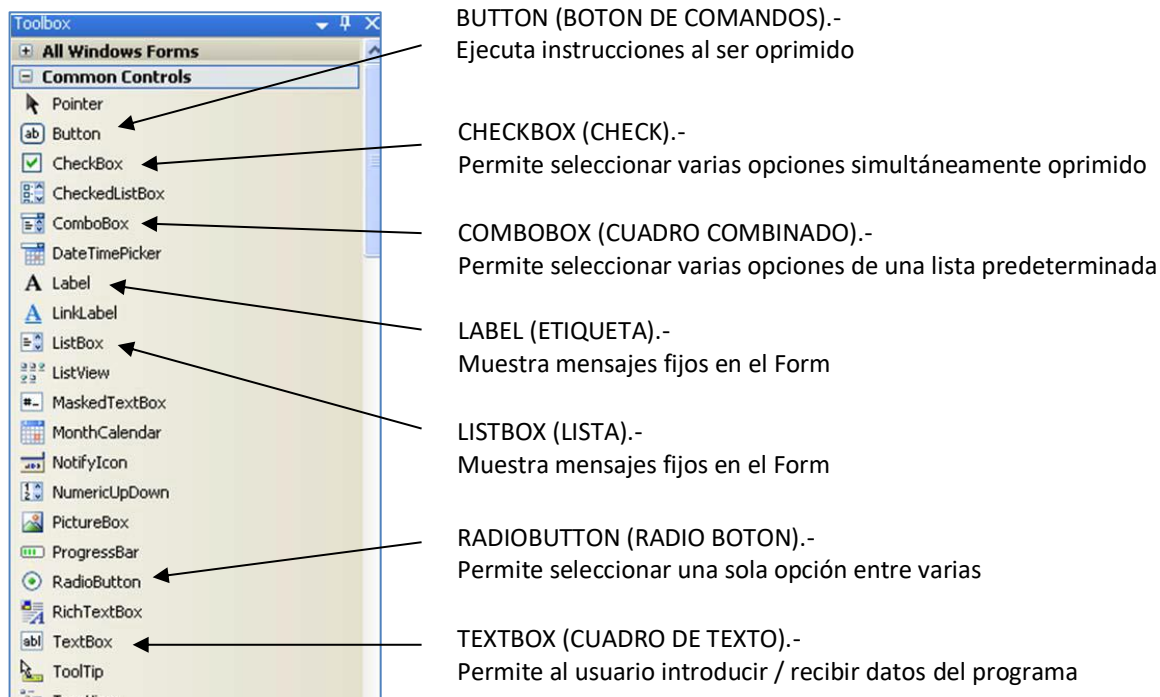
Contenidos:

- ✓ Herramientas (controles) frecuentes de Windows Form C#.
- ✓ Propiedades y eventos de las mismas.
- ✓ Manejo básico de las herramientas.
- ✓ Uso avanzado del control MessageBox en Visual C#.
- ✓ Manejo de botones y opciones mediante la propiedad DialogResult.

HERRAMIENTAS BÁSICAS

Como bien vimos anteriormente, el cuadro de control es aún más amplio que en la imagen de abajo, sin embargo a continuación haremos hincapié sólo en algunas de las herramientas básicas y mayormente usadas en las aplicaciones con formularios de Windows para que podamos avanzar con los contenidos.

(Más adelante profundizaremos en algunas de las siguientes y en otras herramientas necesarias para guías más avanzadas)



Cada control (herramienta) tiene sus propiedades tal como el formulario. Se accede a ellas simplemente colocando un control sobre el form y haciendo clic sobre dicha herramienta. Se pueden arrastrar hacia arriba o abajo para ordenarlas según como creamos mejor (Ej: podemos subir el TextBox cerca del Button, ya que ambos controles son utilizados con mucha frecuencia).

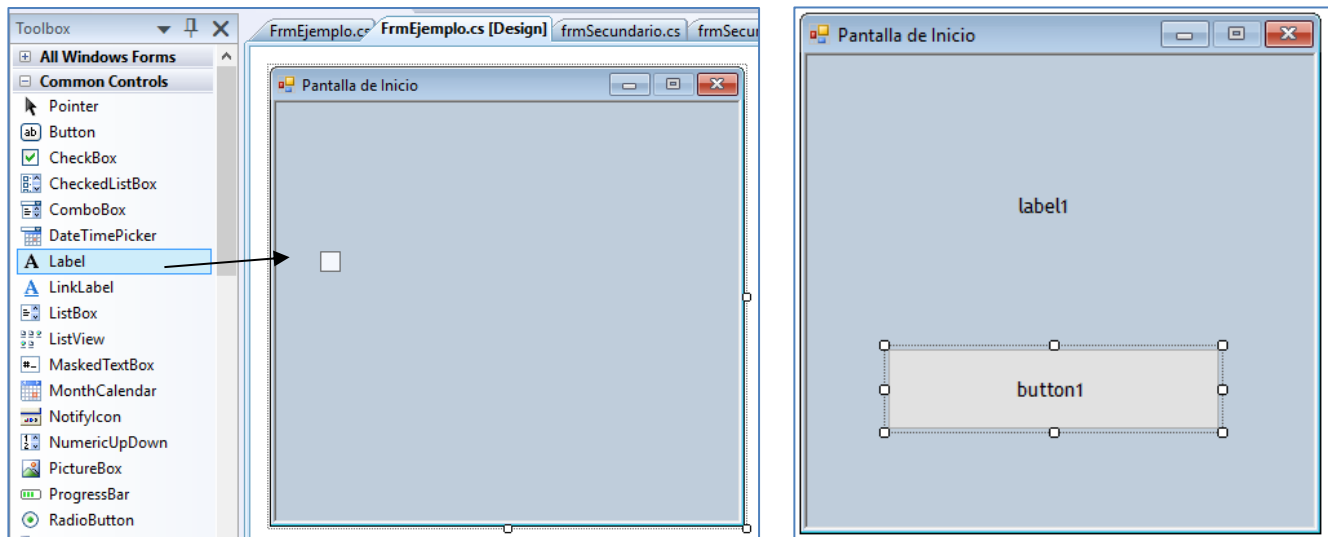
El seguimiento de esta guía es puramente práctico de forma tal que deberán acompañar la lectura con la realización de las actividades.

Pueden crear un nuevo proyecto con el fin de mantener la realización de las guías en orden, crear una copia del proyecto anterior para reutilizar el código y guardar por separado los ejercicios de cada guía o utilizar directamente del proyecto de la guía anterior, el formulario "FrmEjemplo". **Nota: de ser así recordar comentar los eventos MouseMove y DoubleClick para que no interfieran en las siguientes actividades.**

✓ Probando los controles: Label - Button

Realizaremos un programa que muestre en un mensaje dentro de un label y que, al presionar un botón (button) éste cambie de mensaje.

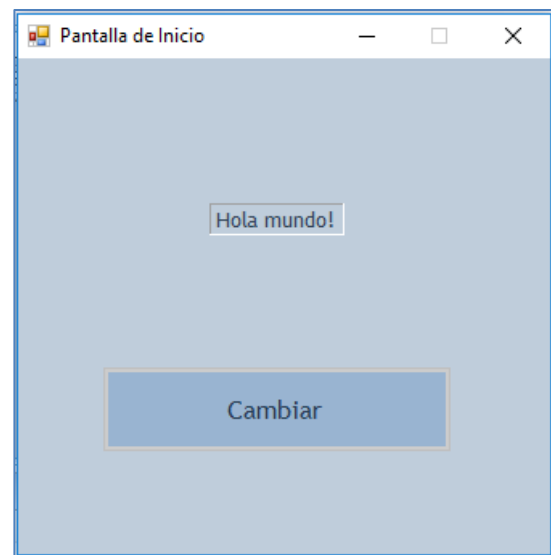
Para comenzar debemos arrastrar un Label y un Button del cuadro de controles y soltar dentro del formulario de la siguiente forma:



A continuación, modificaremos algunas de las propiedades de ambos controles:

(Para hacer esto, debemos hacer clic sobre el control a modificar y luego dirigirnos al cuadro de propiedades)

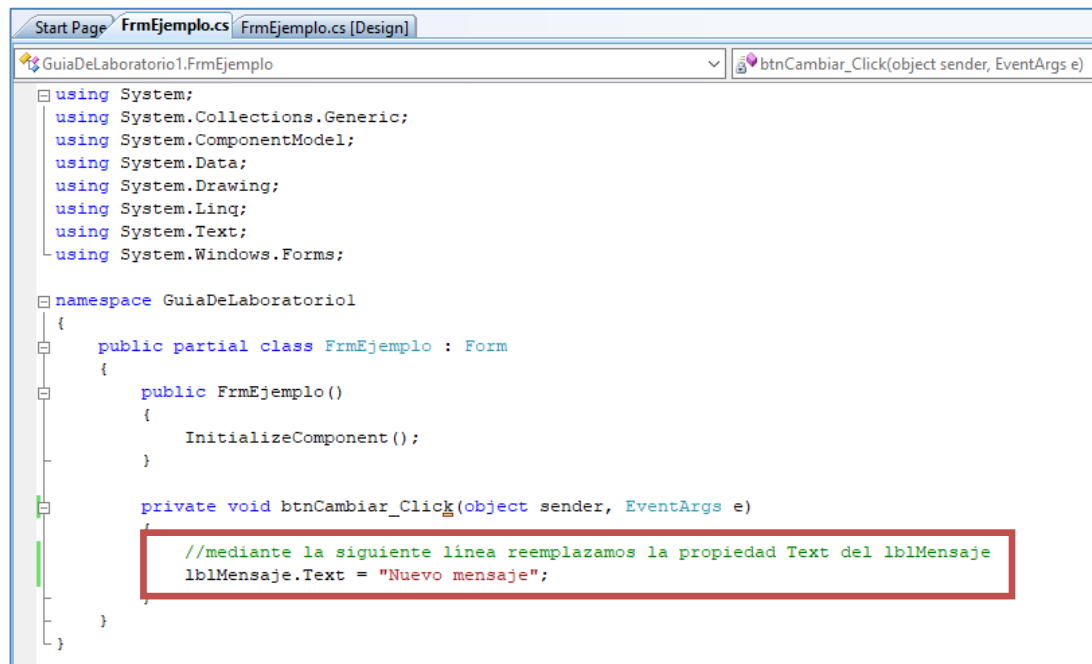
Label1	
Name	lblMensaje
Text	Hola mundo!
AutoSize	True
FormBorderStyle	Fixed3D
Visible	True
Button1	
Name	btnCambiar
Text	&Cambiar
Font: Size	12
BackColor	ActiveCaption



Nótese que a pesar de que exista un solo Button y un solo Label, se les modificó el nombre para que sea más significativo.

El formulario creado, aún no realiza ningún tipo de acción programada. Codificaremos, entonces, el btnCambiar para que cambie el mensaje del lblMensaje.

Hacemos doble clic sobre el btnCambiar, accedemos al código del evento y escribimos la siguiente línea:



```

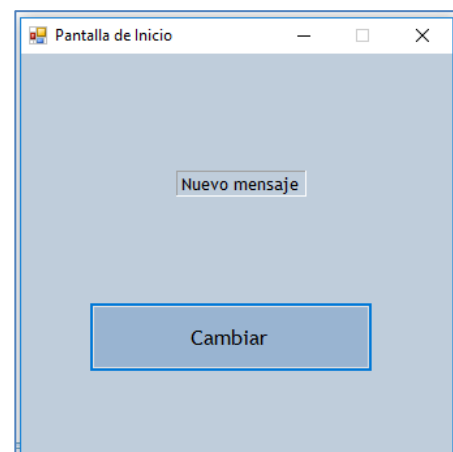
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace GuiaDeLaboratorio1
{
    public partial class FrmEjemplo : Form
    {
        public FrmEjemplo()
        {
            InitializeComponent();
        }

        private void btnCambiar_Click(object sender, EventArgs e)
        {
            //mediante la siguiente línea reemplazamos la propiedad Text del lblMensaje
            lblMensaje.Text = "Nuevo mensaje";
        }
    }
}
    
```

Lo que estamos haciendo es cambiar una de las propiedades de un objeto (en este caso un label o etiqueta), mediante el evento Click de un button. Es decir que, cuando el usuario presione el clic sobre el btnCambiar, el programa se dirigirá a este fragmento de código denominado “función” y realizará lo que nosotros programemos.

De esta forma, al ejecutar el programa se visualizará la pantalla anterior con el mensaje: “Hola mundo!” y al presionar el btnCambiar, el resultado será el siguiente:



✓ Agregamos el control: TextBox

Se solicita un programa tal que permita al usuario escribir un mensaje y que, al presionar un botón, se muestre dentro de un label.

Podemos utilizar el formulario anterior, añadiéndole un TextBox en el cual el usuario pueda escribir el mensaje que desee mostrar en el LblMensaje.

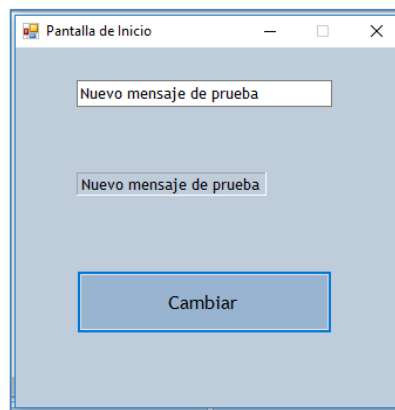
Arrastramos el TextBox al frmEjemplo y modificamos las siguientes propiedades:

textBox1	
Name	txtNuevoMensaje
Text	

En el evento Click de btnCambiar modificamos la línea de código anterior de la siguiente manera:

```
private void btnCambiar_Click(object sender, EventArgs e)
{
    //mediante la siguiente línea reemplazamos la propiedad Text del lblMensaje
    //por lo que se encuentre escrito en el txtNuevoMensaje
    lblMensaje.Text = txtNuevoMensaje.Text;
}
```

El resultado luego de escribir en el cuadro de texto y presionar el botón, debe ser el siguiente:



✓ Probando el control: RadioButton

Modificar el formulario anterior de forma tal que, usando RadioButtons, el usuario pueda elegir entre tres mensajes diferentes para mostrar en el lblMensaje.

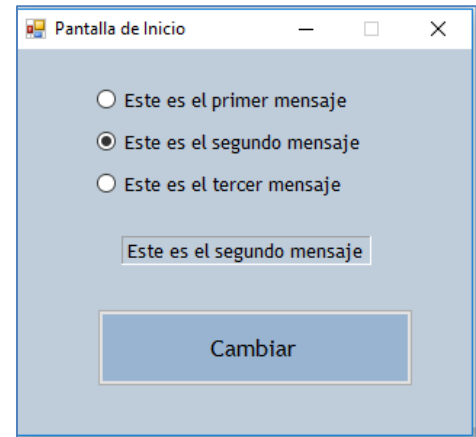
Mantendremos sólo el lblMensaje y el btnCambiar con sus propiedades y agregaremos los siguientes controles:

radioButton1	
Name	rbMensaje1
Text	"Este es el primer mensaje"
radioButton2	
Name	rbMensaje2
Text	"Este es el segundo mensaje"
radioButton3	
Name	rbMensaje3
Text	"Este es el tercer mensaje"

En el evento Click del btnCambiar reemplazaremos lo escrito anteriormente por lo siguiente:

```
//mediante la siguiente línea reemplazamos la
propiedad Text del lblMensaje
//por lo que se encuentre escrito en el radioButton
seleccionado
```

```
if (rbMensaje1.Checked)
    lblMensaje.Text = rbMensaje1.Text;
else if (rbMensaje2.Checked)
    lblMensaje.Text = rbMensaje2.Text;
else
    lblMensaje.Text = rbMensaje3.Text;
```



Como bien se expresa en la imagen inicial de esta guía, los radioButtons son controles que funcionan en conjunto, ya que permiten la selección de uno solo de ellos a la vez.

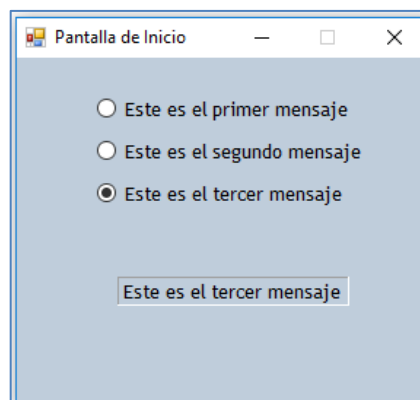
En programas donde dentro del mismo formulario deben coexistir varias listas de “opciones” representadas con este tipo de controles, se utiliza la herramienta contenedora GroupBox (de la cual hablaremos más adelante) para separar los grupos de herramientas a utilizar.

- ❖ Otra opción para realizar esta actividad es reemplazar el texto del label cada vez que se seleccione un radioButton sin necesidad del btnCambiar:

```
private void rbMensaje1_CheckedChanged(object sender, EventArgs e)
{
    lblMensaje.Text = rbMensaje1.Text;
}

private void rbMensaje2_CheckedChanged(object sender, EventArgs e)
{
    lblMensaje.Text = rbMensaje2.Text;
}

private void rbMensaje3_CheckedChanged(object sender, EventArgs e)
{
    lblMensaje.Text = rbMensaje3.Text;
}
```



De esta forma el cambio del mensaje sería automático y no haría falta el uso de un botón. El resultado es el mismo, el cambio de mensaje según la elección de los radioButton.

En el caso de encontrarse desarrollando un programa bajo la plataforma cliente-servidor, asegúrense de tener bien en claro los requerimientos y necesidades del cliente, de no ser así, agotar todas las instancias de consulta posibles, debido a que la elección de dos posibles formas de resolución debe ser funcional a lo solicitado y debe cumplir eficientemente con los parámetros de calidad y funcionamiento del sistema.

✓ Probando el control: ComboBox

En esta actividad la propuesta será igual que la anterior, pero en vez de seleccionar los mensajes representados con RadioButtons, el usuario deberá hacerlo mediante un ComboBox.

El control ComboBox, se usa para mostrar datos en un cuadro combinado desplegable. Si bien su aprovechamiento es mucho mayor al que veremos en esta actividad, recuerden que simplemente estamos conociendo un uso básico de estas herramientas para poder avanzar con los demás contenidos de la materia.

Mantendremos del frmEjemplo, sólo el lblMensaje con sus propiedades y agregaremos los siguientes controles:

comboBox1	
Name	cmbMensajes
Items	Este es el primer mensaje Este es el segundo mensaje Este es el tercer mensaje
FlatStyle	Popup
DropDownStyle	DropDownList

Para agregar los ítems al comboBox, podemos recurrir a diferentes métodos, en primer lugar, como está indicado en la actividad, modificando directamente desde la propiedad Items desde el cuadro de propiedades, donde se desplegará una ventana que nos permitirá escribir el contenido deseado. En segundo lugar, desde la pequeña flecha que se encuentra en la esquina superior derecha del control e ingresando en la opción "Edit Items". Y la tercera puede realizarse mediante código de forma dinámica durante la ejecución del programa, por ejemplo en el form_Load().

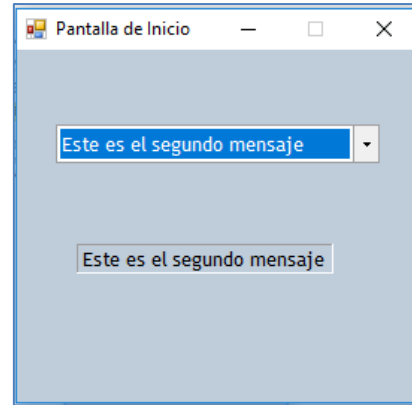
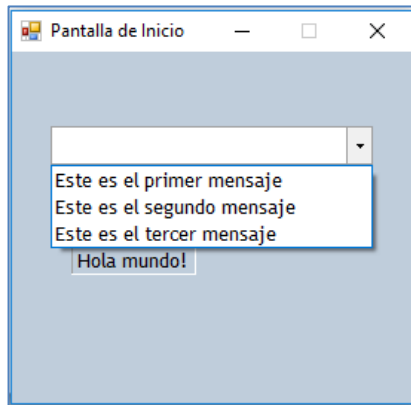
Así como agregamos ítems, también podemos eliminarlos, modificarlos, ordenarlos, pero eso lo veremos más adelante.

Para comenzar con el programa, hacemos doble clic sobre el cmbMensajes, por defecto nos guiará al evento SelectedIndexChanged, donde vamos a programar lo que queremos que suceda si el ítem del combo cambia:

```
private void cmbMensajes_SelectedIndexChanged(object sender, EventArgs e)
{
    /*debemos identificar primero el índice del ítem seleccionado en el
    combo para ello usamos la propiedad SelectedIndex que nos devuelve
    este valor tipo int*/
    int indice = cmbMensajes.SelectedIndex;

    /*para reemplazar el texto del lbl mensaje con el ítem seleccionado
    (indicado con el índice obtenido en la línea anterior) debemos
    convertir este valor a string*/
    lblMensaje.Text = cmbMensajes.Items[indice].ToString();
}
```

Como resultado de esta actividad deberíamos obtener un comboBox que no permita modificar su texto, en un formulario que realice lo siguiente:



✓ Conociendo los controles: ListBox – CheckBox

Se propone modificar el formulario anterior para que aparezcan varios mensajes de forma predeterminada en un listBox y que, al hacer clic sobre alguno de ellos, el mismo reemplace el texto del lblMensaje.

También se solicita que un checkBox habilite y deshabilite la opción de agregar un nuevo mensaje a la lista anterior. textBox y un button, con los cuales se agregarán mensajes al listBox.

Del formulario de la actividad anterior, sólo conservaremos el lblMensaje con sus propiedades y agregaremos los siguientes controles:

listBox1	
Name	lstMensajes
Items	Este es el primer mensaje Este es el segundo mensaje Este es el tercer mensaje
checkBox1	
Name	chkHabilitar
Text	Agregar nuevos mensajes
Checked	False
Button1	
Name	btnAgregar
Text	&Agregar
Font: Size	12
BackColor	ActiveCaption
Visible	False

textBox1	
Name	txtNuevoMensaje
Text	
Visible	False

El control listBox funciona de manera similar a un comboBox, con la diferencia estética en que la lista de elementos se encuentra siempre visible y no en una lista desplegable.

Primero reemplazaremos el texto del lblMensaje con el ítem seleccionado por el usuario del lstMensajes, de la misma forma que lo hicimos con el comboBox:

```
private void lstMensajes_SelectedIndexChanged(object sender, EventArgs e)
{
    int indice = lstMensajes.SelectedIndex;
    lblMensaje.Text = lstMensajes.Items[indice].ToString();
}
```

Al establecer la propiedad Visible = false del txtNuevo Mensaje y el btnAgregar, nos aseguramos que al iniciar el programa, los mismos no serán visibles por el usuario, por lo que, procederemos a programar el chkHabilitar (hacemos doble clic sobre él) para que aparezcan cuando se tildé este control y desaparezcan al destildarlo:

```
private void chkHabilitar_CheckedChanged(object sender, EventArgs e)
{
    if (chkHabilitar.Checked)
    {
        txtNuevoMensaje.Visible = true;
        btnAgregar.Visible = true;
    }
    else
    {
        txtNuevoMensaje.Visible = false;
        btnAgregar.Visible = false;
    }
}
```

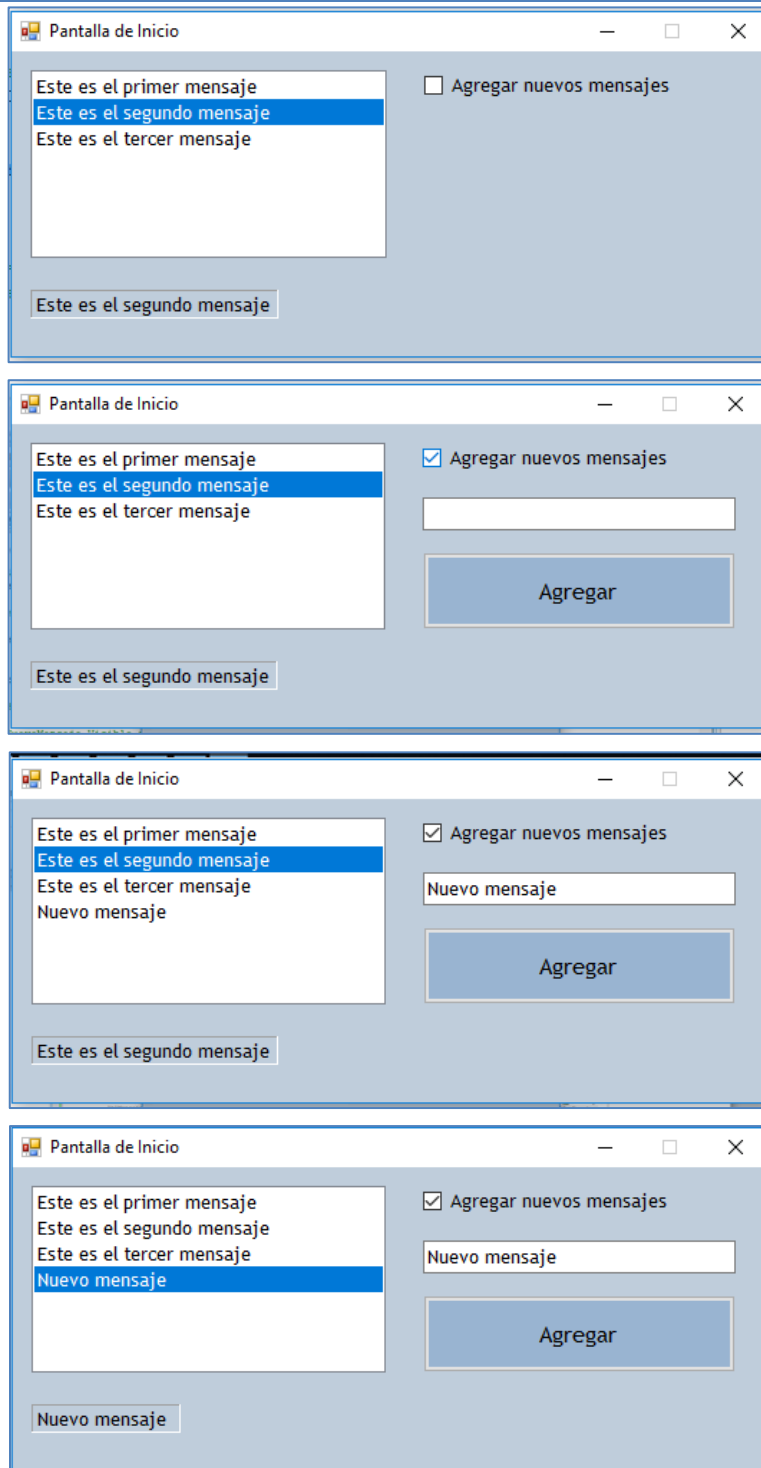
❖ O bien:

```
private void chkHabilitar_CheckedChanged(object sender, EventArgs e)
{
    txtNuevoMensaje.Visible = chkHabilitar.Checked;
    btnAgregar.Visible = chkHabilitar.Checked;
}
```

Una vez habilitados estos dos controles, se deberán poder agregar a la lista los mensajes escritos en el txtNuevoMensaje, para ello accedemos al evento Click del botón Agregar:

```
private void btnAgregar_Click (object sender, EventArgs e)
{
    lstMensajes.Items.Add(txtNuevoMensaje.Text);
}
```

Al finalizar, el resultado será el siguiente:



ESTRUCTURA DEL CONTROL MESSAGEBOX

El control de Windows Forms "MessageBox" muestra un mensaje emergente con acciones.

Se usa para mostrar un mensaje con el texto dado y los botones de acción. También se puede usar el control MessageBox para agregar opciones adicionales, como un título, un icono o botones de ayuda.

Este control utiliza el método "Show()" para mostrar el cuadro de mensaje y recibe una cantidad variada de parámetros que van modificando nuestro MessageBox:

PARÁMETRO (Tipo)	DESCRIPCIÓN
owner (IWin32Window)	Implementación del IWin32Window que será propietario del cuadro de diálogo modal.
Text (String)	Texto que se va a mostrar en el cuadro de mensaje.
Caption (String)	Texto que se va a mostrar en la barra de título del cuadro de mensaje.
Buttons (MessageBoxButtons)	Uno de los valores MessageBoxButtons que especifica qué botones se mostrarán en el cuadro de mensaje.
Icon (MessageBoxIcon)	Uno de los valores de MessageBoxIcon que especifica qué icono se mostrará en el cuadro de mensaje.
defaultButton (MessageBoxDefaultButton)	Uno de los valores de MessageBoxDefaultButton que especifica cuál es el botón predeterminado del cuadro de mensaje.
Options (MessageBoxOptions)	Uno de los valores de MessageBoxOptions que especifica las opciones de pantalla y asociación que se usará para el cuadro de mensaje. Puede transferir 0 si desea usar los valores predeterminados.
displayHelpButton (Boolean)	Es true para mostrar el botón Ayuda; en caso contrario, es false. El valor predeterminado es false.
helpFilePath (String)	Ruta de acceso y nombre del archivo de Ayuda que se va a mostrar cuando el usuario haga clic en el botón Ayuda.
Keyword (String)	Palabra clave de la Ayuda que se va a mostrar cuando el usuario haga clic en botón Ayuda.
Navigator (HelpNavigator)	Uno de los valores de HelpNavigator.
Param (Object)	Identificador numérico del tema de Ayuda que se va a mostrar cuando el usuario haga clic en el botón Ayuda.

Sobrecarga

La forma más fácil de usar MessageBox.Show es escribir "MessageBox", presionar punto y luego seleccionar "Show". A continuación, Visual Studio muestra una ventana emergente con la lista de sobrecarga (combinaciones posibles de parámetros) por las que uno se puede desplazar.

El orden de los parámetros en las llamadas al método MessageBox.Show es importante. El compilador aplica la resolución de sobrecarga para llamar al mejor método en el grupo de métodos.

Opciones de los parámetros

Para un parámetro, como por ejemplo "MessageBoxButtons", se escribe "MessageBoxButtons" y se presiona el punto para ver todas las opciones.

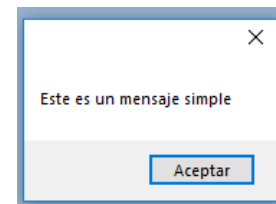
No es necesario crear un nuevo objeto MessageBoxButtons (). Este es un tipo de enumeración, no una clase.

A continuación veremos las formas de combinar estos parámetros y mejorar nuestro MessageBox:

Cuadro de mensaje simple:

El mensaje más sencillo, es el que utilizamos en la primera guía:

```
MessageBox.Show(|
1 of 21 DialogResult MessageBox.Show (string text)
text: The text to display in the message box.
```



```
MessageBox.Show ("Este es un mensaje simple");
```

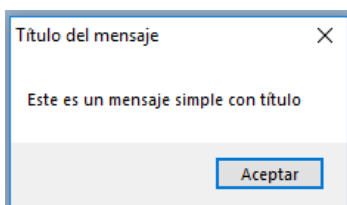
La forma más simple de un cuadro de mensaje es un diálogo con un texto y un botón Aceptar. Cuando haces clic en el botón Aceptar, el cuadro desaparece.

El siguiente fragmento de código crea un cuadro de mensaje simple.

MessageBox con título:

```
MessageBox.Show ("Este es un mensaje simple con título",|
3 of 21 DialogResult MessageBox.Show (string text, string caption)
caption: The text to display in the title bar of the message box.
```

```
MessageBox.Show ("Este es un mensaje simple con título", "Título del mensaje");
```



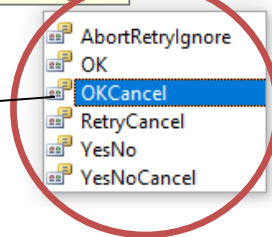
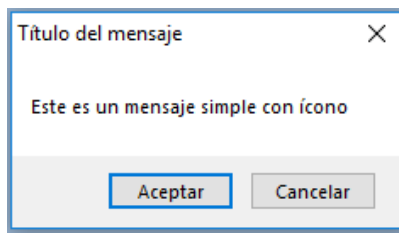
MessageBox con botones:

Un MessageBox puede tener diferentes combinaciones de botones como YesNo y OKCancel. Se accede a ellos mediante "MessageBoxButtons" y tiene los siguientes valores.

```
MessageBox.Show("Desea salir del programa?", "Título del mensaje", MessageBoxButtons.
```

5 of 21 DialogResult MessageBox.Show (string text, string caption, MessageBoxButtons buttons)

buttons: One of the System.Windows.Forms.MessageBoxButtons values that specifies which buttons to display in the message box.

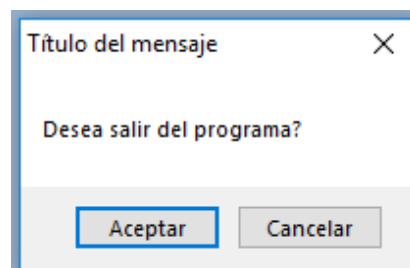


El mensaje anterior es un MessageBox con un título y los botones Aceptar y Cancelar (OKCancel). Este es un MessageBox típico al que se puede llamar cuando se requiere cerrar una aplicación, por ejemplo. Si se hace clic en el botón Sí, la aplicación se cerrará. Para controlar esta decisión del usuario, se utiliza el resultado del método Show, el cual devuelve una enumeración del tipo “**DialogResult**”, para lo que deberíamos modificar el código de la siguiente manera:

```
//creamos y cargamos la variable Resultado que contendrá el valor del
//botón presionado por el usuario
DialogResult Resultado = MessageBox.Show("Desea salir del programa?",
    "Título del mensaje", MessageBoxButtons.OKCancel);

//trabajamos con los posibles valores de la variable Resultado
if (Resultado == DialogResult.OK)
{
    //si el usuario presionó "Aceptar" el programa se cierra
    this.Close();
}

//si el usuario presiona "Cancelar" vuelve al formulario
//sin necesidad de más líneas de código
```



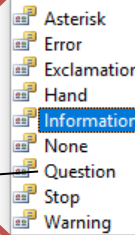
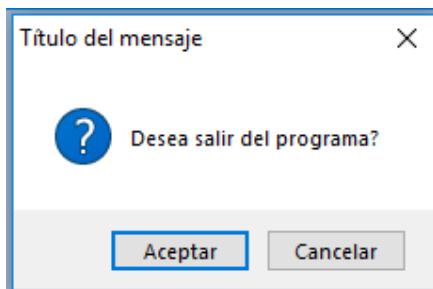
De la misma forma se puede almacenar el resultado de cualquier combinación de botones que asignen al MessageBox.

MessageBox con icono:

Un MessageBox puede mostrar un icono en el cuadro de diálogo. Una enumeración de MessageBoxIcon representa un icono que se mostrará en un MessageBox y tiene los siguientes valores:

```
DialogResult Resultado = MessageBox.Show("Desea salir del programa?",  
    "Título del mensaje", MessageBoxButtons.OKCancel, MessageBoxIcon.Information);
```

7 of 21 DialogResult MessageBox.Show (string text, string caption, MessageBoxButtons buttons, MessageBoxIcon icon)
icon: One of the System.Windows.Forms.MessageBoxIcon values that specifies which icon to display in the message box.



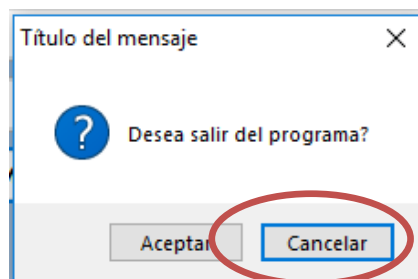
MessageBox con botón predeterminado:

También podemos establecer el botón predeterminado en un MessageBox. Por defecto, el primer botón es el botón seleccionado para acceder inmediatamente presionando la tecla Enter. La enumeración MessageBoxDefaultButton se usa para este propósito y tiene los siguientes tres valores.

- Botón 1
- Button2
- Button3

En el caso de querer establecer como predeterminado el segundo botón, el código sería el siguiente:

```
DialogResult Resultado = MessageBox.Show("Desea salir del programa?", "Título del mensaje",  
    MessageBoxButtons.OKCancel, MessageBoxIcon.Question, MessageBoxDefaultButton.Button2);
```

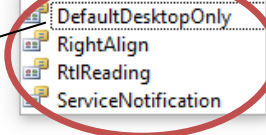
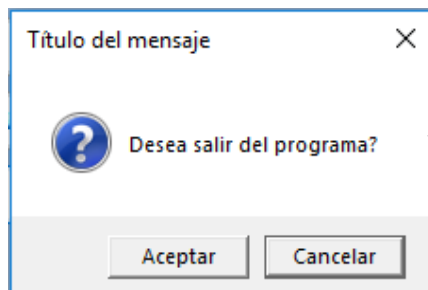


MessageBox con opciones de mensaje:

La enumeración de MessageBoxOptions representa varias opciones y tiene los siguientes valores:

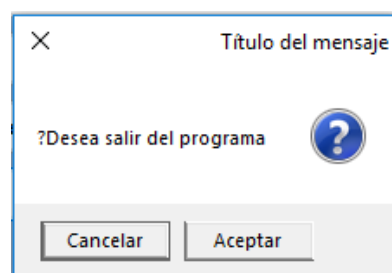
```
MessageBox.Show("Desea salir del programa?", "Titulo del mensaje", MessageBoxButtons.OKCancel,
    MessageBoxIcon.Question, MessageBoxDefaultButton.Button2, MessageBoxOptions.);
```

10 of 21 DialogResult MessageBox.Show (IWin32Window owner, string text, string caption, MessageBoxButtons buttons, MessageBoxIcon icon, **MessageBoxDefaultButton defaultButton**)
defaultButton: One of the System.Windows.Forms.MessageBoxDefaultButton values that specifies the default button for the message box.



En el caso de querer combinar las opciones de aspecto, es posible de la siguiente manera:

```
MessageBox.Show("Desea salir del programa?", "Titulo del mensaje",
    MessageBoxButtons.OKCancel, MessageBoxIcon.Question, MessageBoxDefaultButton.Button2,
    MessageBoxOptions.DefaultDesktopOnly | MessageBoxOptions.RtlReading);
```

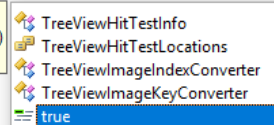


MessageBox con botón de ayuda:

Un MessageBox puede tener un botón adicional llamado botón de Ayuda. Esto es útil cuando necesitamos mostrar un archivo de ayuda. El siguiente fragmento de código crea un MessageBox con un botón de Ayuda.

```
MessageBox.Show("Desea salir del programa?", "Titulo del mensaje", MessageBoxButtons.OKCancel,
    MessageBoxIcon.Question, MessageBoxDefaultButton.Button2, MessageBoxOptions.DefaultDesktopOnly, true
```

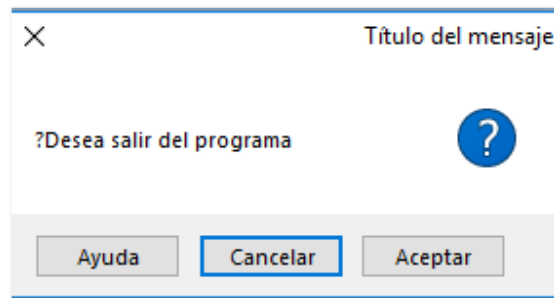
13 of 21 DialogResult MessageBox.Show (string text, string caption, MessageBoxButtons buttons, MessageBoxIcon icon, **MessageBoxDefaultButton defaultButton**, **MessageBoxOptions options**, **bool displayHelpButton**)
Displays a message box with the specified text, caption, buttons, icon, default button, options, and Help button.



En este caso, cambiaremos el valor de MessageBoxOptions a RtlReading, ya que no se permite mostrar un cuadro de mensaje de notificación de servicio con un botón Ayuda, por lo que no se permite la opción DefaultDesktopOnly ni la ServiceNotification.

```
MessageBox.Show("Desea salir del programa?", "Titulo del mensaje",
    MessageBoxButtons.OKCancel, MessageBoxIcon.Question, MessageBoxDefaultButton.Button2,
    MessageBoxOptions.RtlReading, true);
```

En el caso de no querer utilizar una opción de MessageBoxOptions, en el espacio correspondiente a ese parámetro deberá contener el valor '0'.



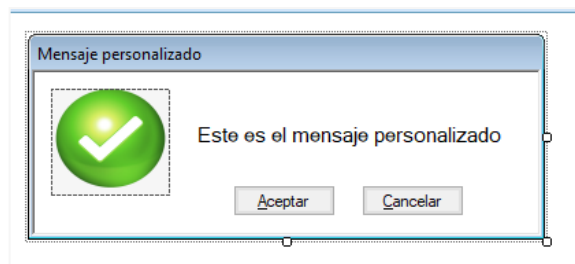
También podemos especificar un archivo de ayuda cuando se hace clic en el botón Ayuda. El siguiente fragmento de código hace referencia a un archivo de ayuda.

```
1. DialogResult result = MessageBox.Show (mensaje, título,
2. botones, MessageBoxIcon.Question, MessageBoxDefaultButton.Button1,
   0, "helpfile.chm" );
```

OTRA OPCIÓN PARA UTILIZAR DIALOGRESULT

En el caso de requerir un MessageBox personalizado, existe la forma de crearlo mediante un formulario nuevo.

Creamos un formulario y personalizamos su interfaz de la forma deseada:



Desde el formulario de origen, en el evento que se llamará al mensaje personalizado, colocamos las siguientes líneas de código:

(En este caso se hizo en el evento Click de un botón)

```
private void btnFormulario_Click(object sender, EventArgs e)
{
    Form FormMensaje = new frmMessageBox();

    //creamos una variable del tipo DialogResult que
    //recibirá el resultado de lo seleccionado por el usuario
    //en el formulario mensaje
    DialogResult Result = FormMensaje.ShowDialog();

    //control de la variable
    if (Result == DialogResult.OK)
        btnFormulario.Text = "OK";
    else
        btnFormulario.Text = "Cancel";
}
```

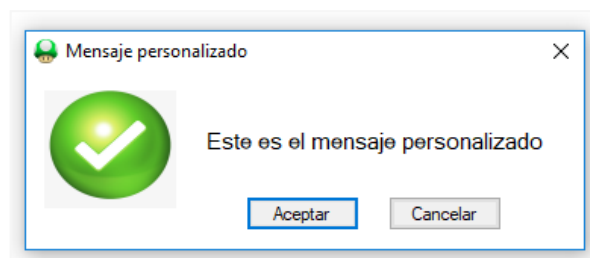
Lo que hacemos en estas líneas es abrir un formulario con el método “ShowDialog” para devolver como resultado la elección del usuario.

Para poder recibir este dato, primero debemos programar los botones de nuestro nuevo formulario de mensaje:

```
private void btnAceptar_Click(object sender, EventArgs e)
{
    //si el usuario presiona este botón, la variable "Result"
    //del formulario de origen, recibirá como dato "DialogResult.OK"
    this.DialogResult = DialogResult.OK;
}

private void btnCancelar_Click(object sender, EventArgs e)
{
    //si el usuario presiona este botón, la variable "Result"
    //del formulario de origen, recibirá como dato "DialogResult.Cancel"
    this.DialogResult = DialogResult.Cancel;
}
```

Si bien no es muy diferente a la forma de programar los mensajes como vimos desde el inicio de esta guía, el resultado de esto, es un MessageBox con el tamaño, el ícono, la imagen, los botones y los nombres de los mismos que uno desee.



4.1. Ejercicio práctico (no entregable): Realizar un programa que cree una lista de alumnos, registrando nombre y apellido, pero también si es alumno regular o libre. Ejemplos: “Galeano Esteban - R”, “Campos Romina - L”.

Solo el nombre y apellido pueden ser ingresados mediante un textBox, la condición de libre o regular deberán ingresarla con otra herramienta y luego concatenar el valor al nombre y apellido **antes** de agregar el ítem a la lista.

Cada ítem se agregará presionando un botón, como en el paso a paso realizado en esta guía, pero en este ejercicio, cuando el usuario presione el botón para agregar al alumno, se le consultará mediante un MessageBox, utilizando DialogResult, si está seguro de agregarlo a la lista. De aceptar, se agrega tal como se indica, pero de cancelar la acción, se volverá al formulario, haciendo foco en el primer textBox (Nombre o Apellido). (Ej: `textBox1.Focus();`)

Sugerencia para una buena práctica de ahora en más: cada objeto dentro del formulario, posee una propiedad llamada “**TabIndex**”, la cual indica, por defecto, el orden en que fueron creados, pero esto se puede modificar con el fin de que, si el usuario debe interactuar con varios de estos objetos, por ejemplo completar un formulario, seleccionar opciones, etc., pueda ir avanzando en orden, solo presionando la tecla Tab. Es decir, si el orden para completar un formulario es Nombre, Apellido, Domicilio, Teléfono, los números de la propiedad TabIndex deben ser seguidos, por ejemplo: `txtNombre.TabIndex = 2`, `txtApellido.TabIndex = 3`, `txtDomicilio.TabIndex = 4`, `txtTeléfono.TabIndex = 5`. De modo que, al completar el nombre, solo se deberá presionar la tecla Tab y el cursor se posicionará en el txtApellido, lo mismo para avanzar al txtDomicilio y al txtTeléfono. Este valor se puede asignar desde la vista de diseño. No hace falta hacerlo mediante código.

