Tema 2: Menús y Cuadros de Diálogo

¿Qué aprenderás?

- Conocer y utilizar los cuadros de diálogo disponibles en .NET
- Crear nuevos formularios y cuadros de diálogo
- Uso de menús y barras de herramientas.

¿Sabías que…?

- Los menús permiten establecer atajos de teclado que agilizan enormemente la interfaz.

2.1. Uso de formularios

2.1.1. Diálogos predefinidos

En muchos casos no tenemos suficiente con un formulario para gestionar todas las funcionalidades de un programa. En algunos casos hay que utilizar múltiples formularios. El entorno de Visual Studio .NET dispone de un conjunto de formularios prefabricados que gestionan algunas necesidades frecuentes (como por ejemplo, cómo abrir y guardar ficheros, imprimir datos, escoger fondo o colores, etc.) y que nos pueden ahorrar trabajo. En otras ocasiones deberemos diseñar un formulario desde cero. Ahora vamos a trabajar en los formularios predefinidos que tenemos a nuestra disposición.

2.1.2. Mensaje de Alerta

Uno de los formularios más necesarios son estas pequeñas ventanas emergentes, que contienen un breve mensaje para el usuario. Estas ventanas se pueden generar con una simple función **MessageBox** que recibe como parámetros el texto que ha de mostrar y el título que mostrará.

Generamos un proyecto con un botón y al evento clic del botón le ponemos la siguiente línea de código.

```
MessageBox.Show("Mensaje de aviso","ALERTA",MessageBoxButtons.OK,
MessageBoxIcon.Warning);
```

El resultado será la ventana siguiente:

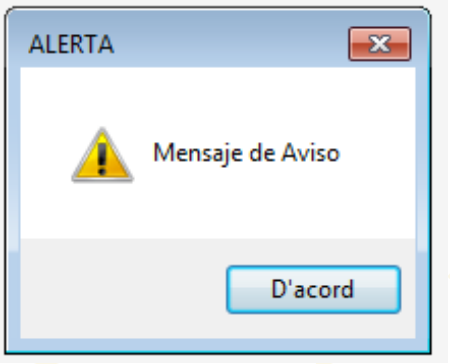


Figura: ventana de alerta

Cuando ejecutamos una función **MessageBox**, el programa se para hasta que el usuario cierra la ventana de alerta.

En algunos casos nos interesa que el usuario escoja entre distintas opciones; esto lo podemos lograr usando una combinación de botones que contenga las opciones que vamos a ofrecer al usuario.

Modificamos la línea de código anterior por la siguiente:

```
MessageBox.Show("Cerrar aplicación?","SALIR",MessageBoxButtons.YesNo,
MessageBoxIcon.Question);
```

Ahora la ventana que se muestra es la siguiente:

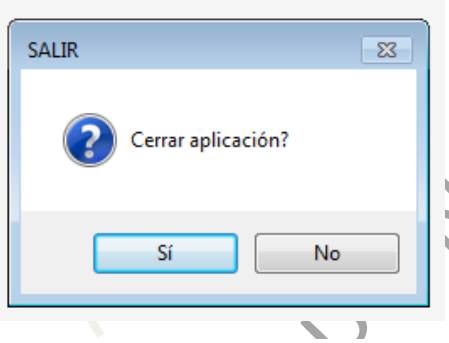


Figura: ventana de alerta con opciones

Pero fijémonos que tanto si respondemos "sí" como si respondemos "no" el programa no hace nada. Lo que hemos de hacer es recoger la respuesta del usuario del programa principal, y esto lo podemos hacer con una simple sentencia If:

Las ventanas de diálogo devuelven el botón con el que han sido cerradas en un enumerable llamado **DialogResult** por lo que la escribirlo nos ofrece una lista de sus valores.

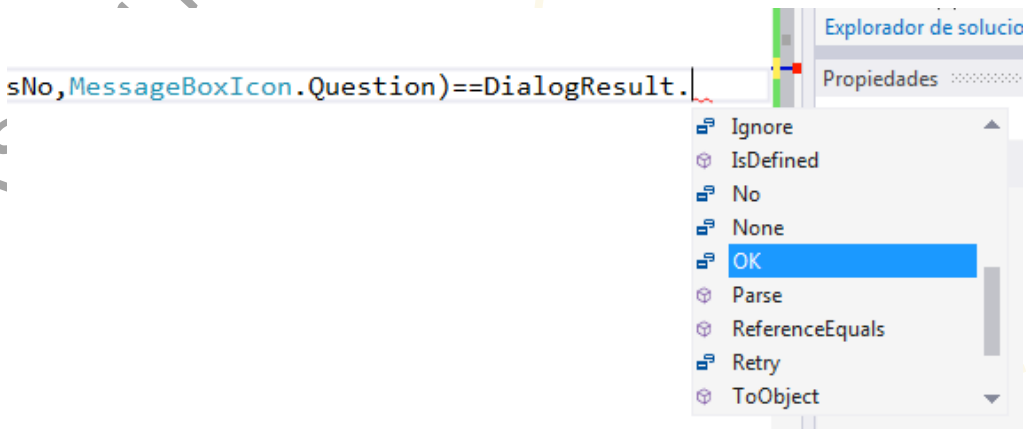


Figura: Respuestas de una ventana de alerta con opciones

El programa quedará de esta manera:

```
if (MessageBox.Show("Cerrar aplicación?","SALIR",MessageBoxButtons.YesNo,
MessageBoxIcon.Question)==DialogResult.Yes)
{
  this.Close()
}
```

2.1.3. Diálogo de apertura de fichero

Los próximos formularios que vamos a ver los podemos encontrar en el cuadro de herramientas en el apartado llamado "**Cuadros de Diálogo**". El primero que vamos a estudiar es el que permitirá al usuario escoger uno de los ficheros del disco para abrir.

Generamos un nuevo proyecto y colocamos un botón que ponga **Abrir**. A continuación, de la sección de cuadros de diálogo de los controles, escogemos un elemento del tipo **OpenFileDialog** y lo ponemos en el formulario:

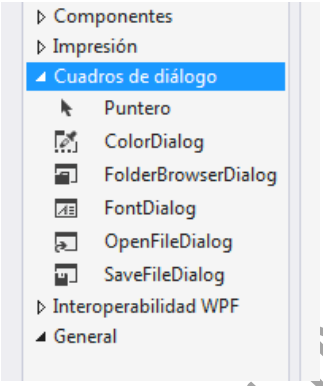


Figura: Cuadros de Diálogo

Notaremos que este control no aparece en el formulario, sino que se muestra en la parte inferior del entorno de programación.

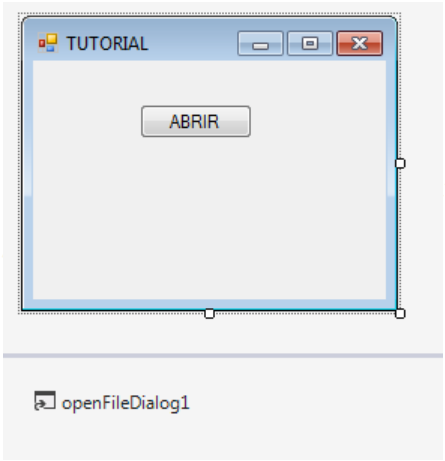


Figura: Control OpenFileDialog

Para mostrar el cuadro de diálogo hemos de poner la siguiente instrucción en el evento clic del botón Abrir:

```
openFileDialog1.ShowDialog();
```

Ahora sí que se ejecutará el programa, veremos la típica ventana de abrir fichero y podremos acceder a todo nuestro ordenador. Pero de momento, el programa no hace nada cuando escogemos un fichero.

Aunque el programa no haga nada, lo más importante es saber qué fichero o ficheros han sido seleccionados en el diálogo. Esta información la tenemos en el atributo **FileName** con la ruta absoluta de cada uno de los ficheros. Añadimos una instrucción a continuación de la anterior para mostrar el fichero escogido en una caja de mensaje:

```
openFileDialog1.ShowDialog();
MessageBox.Show(openFileDialog1.FileName);
```

Es importante fijarse en que el diálogo no permite seleccionar más de un fichero. Si queremos habilitar la posibilidad de seleccionar múltiples ficheros, hay que modificar el valor de la propiedad **MultiSelect** del **OpenFileDialog** de **False** a **True**. Cuando hacemos esto, ya no podemos utilizar el atributo **FileName** porque ahora ya podemos tener más de un nombre y en lugar de esta, utilizaremos **FileNames** que es un **String[]** que contiene las rutas de los ficheros escogidos. Activamos la selección múltiple en nuestro proyecto y modificamos el código del botón por el siguiente:

```
openFileDialog1.ShowDialog();
foreach ( String fichero in openFileDialog1.FileNames)
{
  MessageBox.Show(fichero);
}
```

Vamos a hacer algo con el fichero escogido. A nuestro proyecto le añadimos un control **PictureBox**. Modificaremos el código del botón para que se muestre el fichero escogido en la caja de imagen:

```
openFileDialog1.ShowDialog();

pictureBox1.Image = Bitmap.FromFile(openFileDialog1.FileName);
```

El programa funciona perfectamente, siempre que el fichero escogido sea una imagen, en caso contrario, se genera una excepción. Esta la podemos solucionar con un bloc **Try/Catch**.

```
openFileDialog1.ShowDialog();
try
{
  pictureBox1.Image = Bitmap.FromFile(OpenFileDialog1.FileName);
}
catch
{
  MessageBox.Show("Imagen incorrecta",”ERROR",MessageBoxButtons.Ok,
MessageBoxIcon.Error);
}
```

De todas formas, podemos mejorar el programa si en el cuadro de diálogo solo aparecen imágenes. Para hacer esto, utilizaremos la propiedad **Filter** del cuadro de diálogo. Los filtros tienen la siguiente forma:

<Nombre del filtro>|<extensiones a filtrar>

Para mostrar solo las imágenes aplicamos el siguiente filtro:

```
Imagenes|*.jpg
```

Ahora el aspecto del cuadro de diálogo será el siguiente:

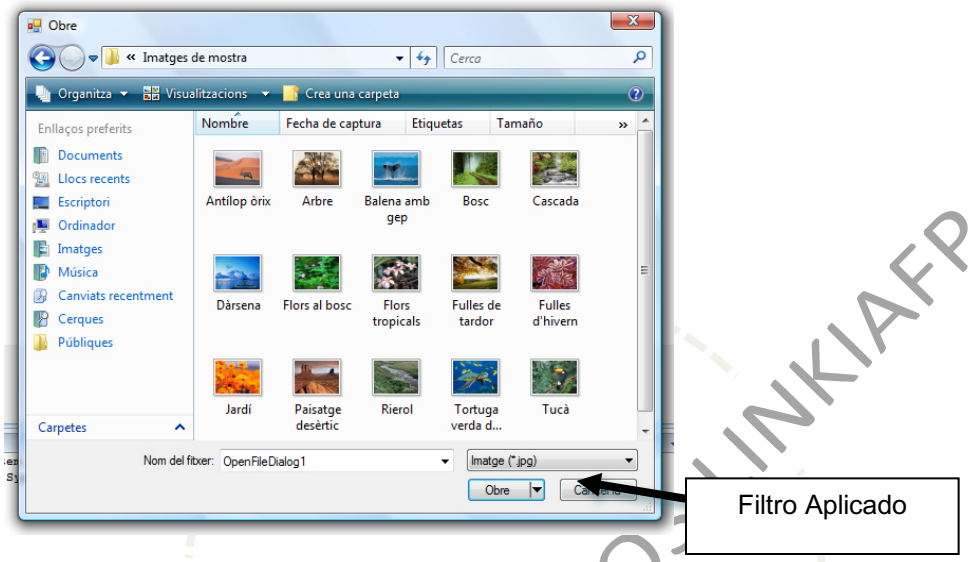


Figura: Control OpenFileDialog

El problema es que solo se visualizan los ficheros con la extensión jpg. Si quisiéramos visualizar más extensiones, podemos modificar el filtro y especificar todas las extensiones que queramos separadas por un punto y coma.

```
Imagenes|*.jpg;*.gif;*.png
```

Ahora el filtro aplicado muestra las tres extensiones:

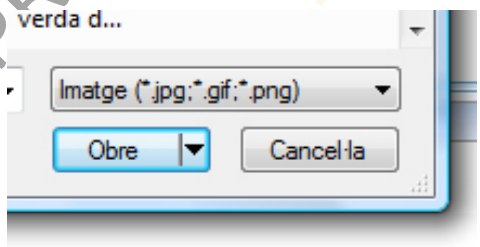


Figura: Filtro de extensiones

Si queremos aplicar más de un filtro, solo debemos encadenarlos todos juntos separados por una barra vertical:

```
Imagenes|*.jpg;*.gif;*.png|Todos|*.*
```

Ahora el usuario puede cambiar el filtro aplicado en el cuadro de diálogo para mostrar imágenes o bien todos los ficheros:

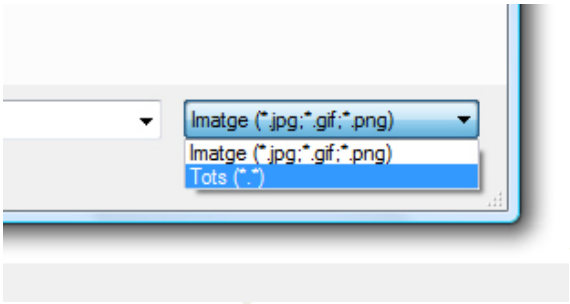


Figura: Filtro de extensiones

El resto de los cuadros de diálogo que tenemos disponibles funcionan de la misma manera. Se muestran con el método **ShowDialog** y una vez cerrados, podemos acceder a la información deseada a través de las propiedades del cuadro de diálogo.

2.1.4 Creación de formularios

En muchos casos, las ventanas prefabricadas no nos sirven. En estos casos, hemos de generar un nuevo formulario y programarlo desde cero nosotros mismos. Para poder generar un nuevo formulario para nuestra aplicación, lo hacemos desde el menú (***Menú Proyecto → Agregar Windows Forms...***) y ponemos el nombre que queramos al formulario.

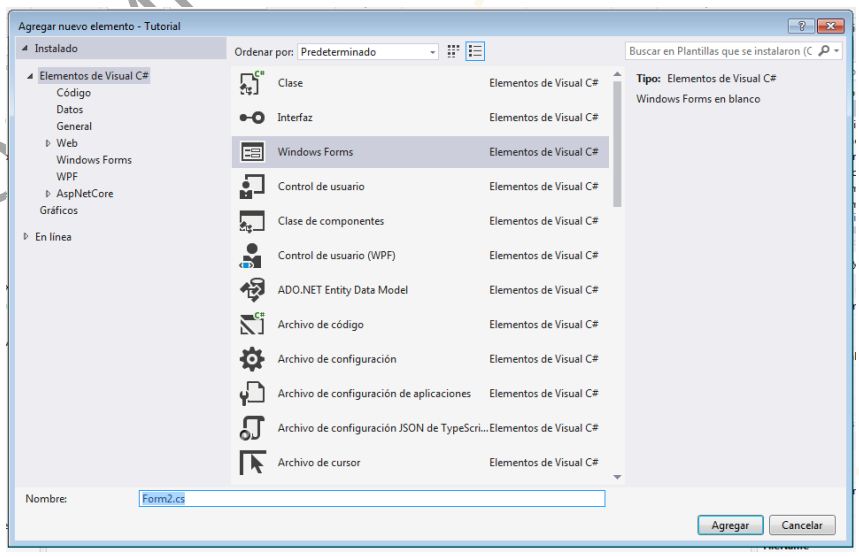


Figura: Formulario Nuevo

Con esto se genera un formulario nuevo y en blanco como el primero.

Podemos observar el nuevo formulario en la zona **Explorador de Soluciones** del entorno de programación.

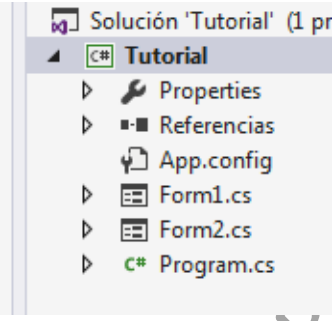


Figura: Explorador de ficheros

Ahora debemos implementar el nuevo formulario con las herramientas estudiadas hasta el momento. Una vez diseñado nuestro formulario, necesitamos visualizarlo. Para conseguirlo lo podemos hacer de dos formas distintas: forma **Modal** o forma **No Modal**. Cuando decidimos utilizar un formulario Modal, este se coloca en primer plano y no podemos cambiar el formulario activo ni minimizarlo. Lo único que podemos hacer es cerrarlo.

En cambio, si decidimos utilizar la forma No Modal, el nuevo formulario se comporta como una segunda ventana independiente que podemos minimizar o cerrar cuando queramos. Para practicar esto, generamos un nuevo proyecto y añadimos un botón. Entonces generamos un segundo formulario nuevo. Utilizamos el botón para visualizar este segundo formulario:

Forma Modal:

```
Form2 f = new Form2();
f.ShowDialog();
```

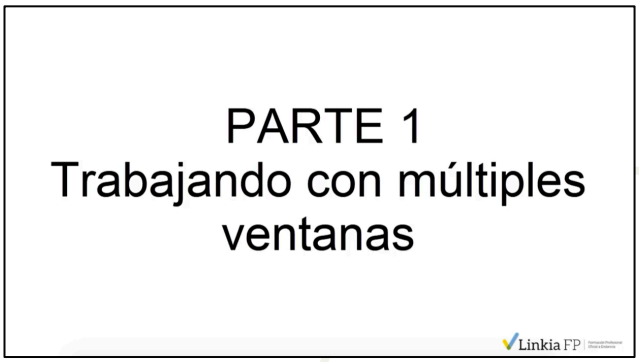
Forma No Modal:

```
Form2 f = new Form2();
f.Show();
```

Es importante destacar que, si utilizamos la forma Modal, el programa se detiene hasta que cerremos la nueva ventana que aparece, en cambio, si utilizamos la otra forma, la ventana principal continúa ejecutándose normalmente. Esto lo podemos comprobar añadiendo la siguiente instrucción a continuación de los métodos **Show** y **ShowDialog**:

```
MessaegBox.Show("Nueva Linea");
```

Comprobamos que, en el caso Modal, no visualizamos la alerta hasta que el segundo formulario se cierra, mientras que, en el caso No Modal, la ventana aparece inmediatamente después de mostrar el segundo formulario (prueba de que el primer formulario continúa su ejecución).



Video: Uso de múltiples ventanas (parte 1)

Una vez ya sabemos mostrar el formulario, lo que tenemos que aprender es cómo proporcionar datos al segundo formulario desde el primero y a devolver datos del segundo al primero.

Si tenemos en cuenta que los formularios son clases como cualquier otra, podemos solucionar el problema simplemente modificando los constructores y las llamadas para pasar parámetros:

From2:

```
Form1 f1;
public Form2(Form1 f)
{
  InitializeComponent();
  this.f1 = f;
}
private void Form2_Shown(object sender, EventArgs e)
{
  label1.Text = f1.cadena;
}
```

Form1:

```
public String cadena;
private void Button1_Click(object sender, EventArgs e )
{
  cadena = "Hola soy From1";
```

```
  Form2 f = new Form2(this);
  f.ShowDialog();
}
```

Si queremos pasar datos al primer formulario, lo hacemos de la misma forma, pero es importante destacar, que las variables se tienen que declarar siempre en el primer formulario porque este es el único que no se va a cerrar, ya que cuando se hace, la aplicación se cierra.

Lo único que nos queda por saber es cómo cambiar el formulario de inicio. Por ejemplo, si en nuestro caso, quisiéramos que el **Form2** fuera el primero en visualizarse, lo podríamos hacer modificando el programa principal (que no es el Form1). Editar el fichero Program.cs que podemos ver en el explorador de archivos de nuestra solución.

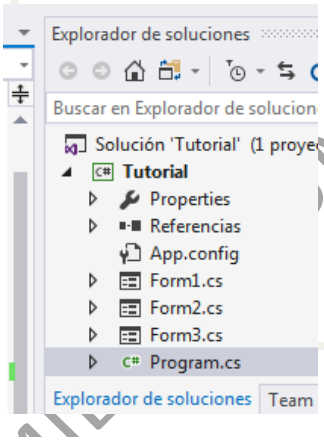
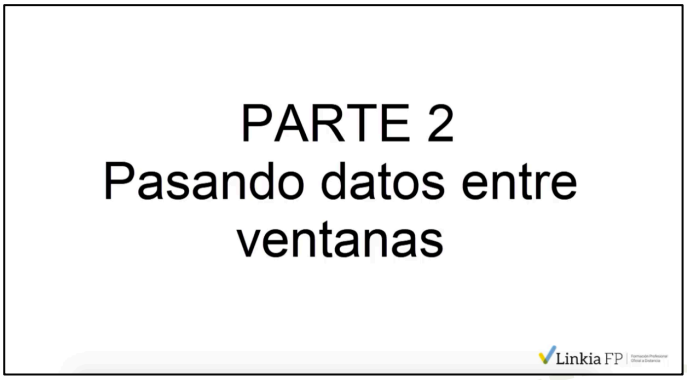


Figura: Fichero inicial del proyecto

En el podemos encontrar el método **Main**() que es el primero que se ejecuta al lanzar la aplicación. Observar que hay una línea que contiene una llamada al método **Run** del objeto **Application**.

```
static void Main()
{
  Application.EnableVisualStyles();
  Application.SetCompatibleTextRenderingDefault(false);
  Application.Run(new Form1());
}
```

Ésta es la encargada de lanzar el formulario inicial. Sólo tenemos que cambiar Form1 por el formulario por el que queremos.



Video: Uso de múltiples ventanas (parte 2)

2.2. Menús y barras de herramientas

2.2.1. Menús

El uso de menús y barras de herramientas en las aplicaciones es una forma de ahorrar espacio en nuestros formularios, así como para organizar las funcionalidades de nuestra aplicación de forma intuitiva y accesible. Prácticamente todas las aplicaciones disponen de una o más barras de menú. Todos los controles que vamos a trabajar en este apartado, los podemos encontrar en la sección **Menús y barras de herramientas** del cuadro de controles.

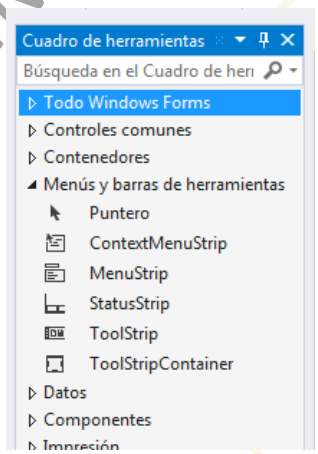


Figura: Controles de Menús y barras de herramientas

Empezaremos trabajando con los menús. Generamos un proyecto y añadimos un control del tipo **MenuStrip** al formulario. El aspecto que ahora presenta el formulario es el siguiente:

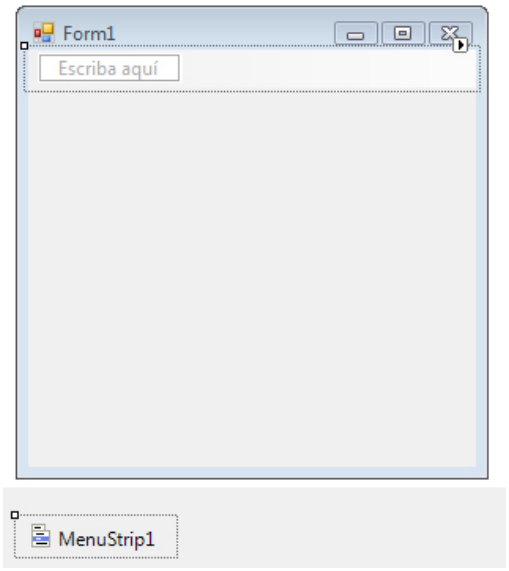


Figura: Nuevo menú

Para generar el menú, lo único que hemos de hacer es ir escribiendo el nombre a cada una de las opciones que queremos, generando la estructura de grupos y subgrupos que deseemos.

Empezamos generando un grupo llamado General donde ponemos las opciones Abrir y Salir; el aspecto que debería tener es el siguiente:

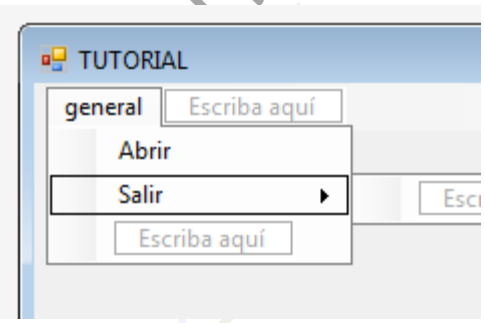


Figura: Añadiendo opciones (I)

Ahora si ejecutamos nuestra aplicación veremos que el menú se despliega de forma correcta, pero cuando seleccionamos alguna de las opciones no sucede nada. Lo que nos toca hacer ahora es escribir el código asociado a cada una de las opciones del menú. Para conseguir esto, solo hemos de hacer doble clic encima de una de ellas para que el entorno de programación nos genere el evento correspondiente.

```
private void salirToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.Close();
}

private void abrirToolStripMenuItem_Click(object sender, EventArgs e)
{
    openFileDialog1.ShowDialog();
}
```

Notar que se tiene que añadir un cuadro de diálogo **OpenFileDialog** para simular la opción Abrir.

Ahora cuando ejecutamos la aplicación, las dos opciones del menú hacen lo que se supone que deben hacer.

En estos momentos, ya tenemos la capacidad de generar menús para nuestras aplicaciones, pero podemos mejorar su funcionalidad. Si nos hemos fijado en las aplicaciones donde hay menús, podemos acceder a TODAS las opciones de los menús combinando la tecla **ALT** con una o más teclas para crear un atajo. La tecla que hemos de combinar con ALT, para activar cada una de las opciones, se puede visualizar cuando pulsamos la tecla **ALT** (es la letra subrayada en cada una de las opciones).

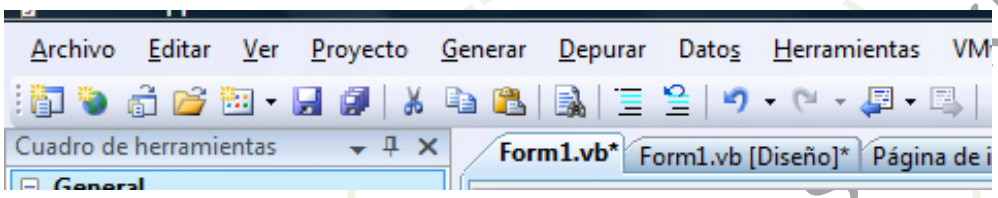


Figura: Atajo del teclado

Podemos comprobar que cuando pulsamos **ALT+A** en el entorno de programación se despliega el menú Archivo. Para aplicar esta funcionalidad en nuestra barra de menús, lo que tenemos que hacer es modificar el texto de las opciones del menú añadiendo el carácter **"&"** delante de la letra que queremos que sea el atajo. En nuestro caso, haremos que la letra clave de la opción general sea la **"G"**, por lo tanto, escribiremos **&General** como texto de la opción. Observaremos que en el momento de aceptar el cambio de texto la letra marcada queda subrayada indicando su función de dirección de teclado.

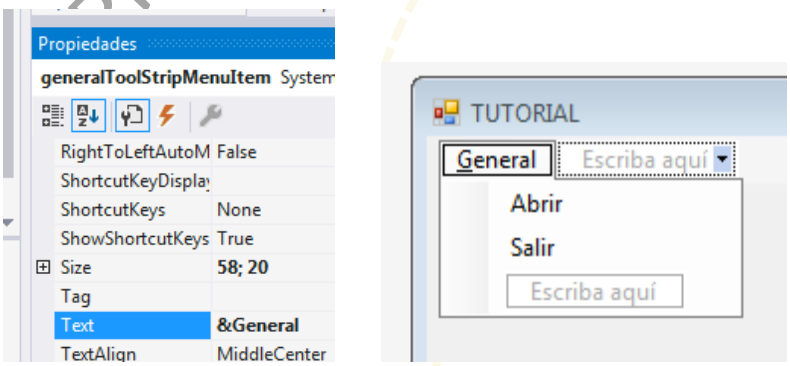


Figura: Añadiendo opciones (II)

Ahora repetimos el proceso para las opciones Abrir (seleccionaremos la A) y Salir (escogemos la S). El aspecto del menú será el que se muestra en esta imagen:

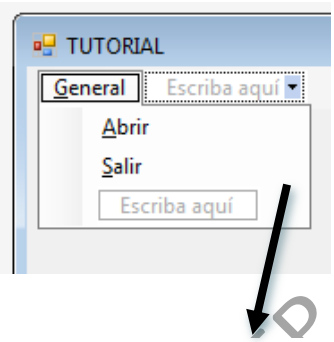


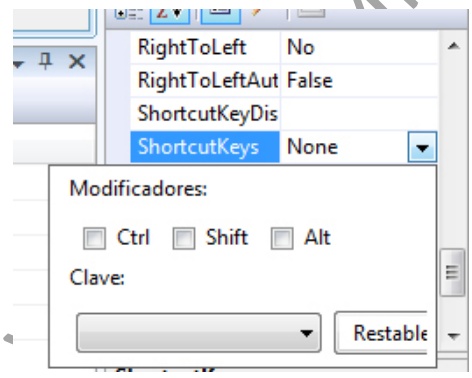
Figura 72: Añadiendo opciones (III)

Comprobamos que cuando ejecutamos la aplicación, podemos activar las opciones combinando la tecla ALT con las letras subrayadas; pero notad que primero hay que activar **ALT+G** para desplegar el menú, y después, la combinación correspondiente a la opción escogida. Esto significa que hemos de utilizar dos atajos consecutivos.

Hay opciones que resultan muy frecuentes en una aplicación, en estos casos, quizá nos interesa generar un atajo "***directo***" para esta opción. De esta manera no tenemos que activar primero el menú y después la opción. En este caso, podemos utilizar la propiedad **ShortcutKeys** de la opción. Para hacer esto, utilizaremos el inspector de propiedades para editar la propiedad que actualmente tiene el valor "**none**" (ninguno en inglés). Activamos el desplegable y veremos una ventana que nos guía en el proceso de generar el atajo del teclado.

Crearemos un atajo directo a la opción Salir con la combinación **Ctrl+S**.

Figura: Añadiendo opciones (IV)



Comprobaremos que en el momento que generamos un atajo directo a una opción, la combinación de las teclas escogidas aparece escrita al lado de la opción en el menú; tanto en la vista del diseño como en ejecución.

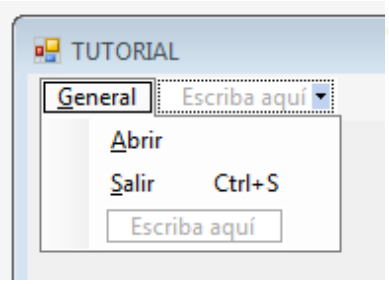


Figura: Añadiendo opciones (V)

Ahora si ejecutamos nuestra aplicación comprobaremos, que tenemos suficiente con pulsar Ctrl+S para activar la opción Salir y por tanto cerrar la aplicación.

Otro detalle que podemos mejorar en nuestros menús es el de añadir una imagen a las opciones del menú. Para poder hacerlo, es suficiente con editar la propiedad **Image** de la opción y escoger una imagen del disco. En nuestro caso podemos dejar el menú tal y como se muestra en la imagen:

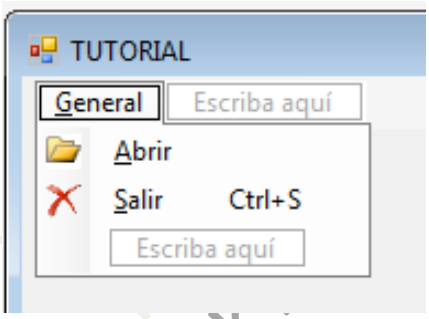


Figura: Añadiendo opciones (VI)

Ya podemos generar una barra de menús como queramos.

2.2.2. Menús contextuales

Los menús contextuales son aquellos que se visualizan utilizando el botón derecho (o secundario) del ratón sobre un determinado elemento. Para generar un menú contextual, añadimos a nuestro proyecto un elemento del tipo **ContextMenuStrip**. El aspecto del proyecto es el que se muestra en la siguiente imagen:

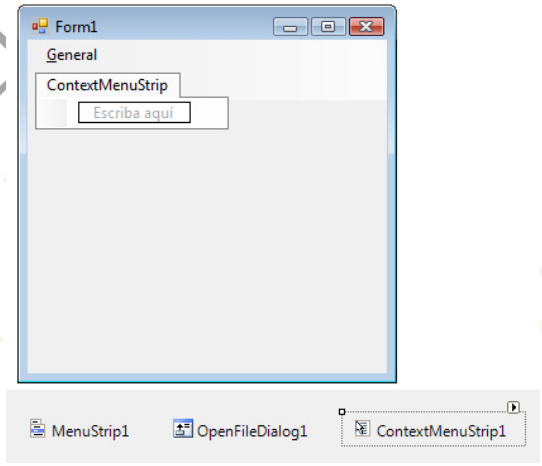


Figura: Menú contextual

Para diseñar un menú de contexto hay que seguir las mismas reglas que para un menú. Fijaos que si hacemos clic fuera del menú contextual este desaparece. Para continuar editándolo, es suficiente con seleccionar el objeto correspondiente de la zona inferior de la ventana de diseño y este aparece de nuevo en la zona central del formulario. Añade al menú de contexto las mismas opciones que el menú general.

Si ahora ejecutamos la aplicación, nos daremos cuenta, que si pulsamos el botón secundario del ratón no aparece el menú de contexto que hemos generado.

Esto es así, porque, al contrario que en los menús generales, los menús de contexto se tienen que asociar a un control. Para asociar un menú de contexto a un control hemos de editar las propiedades del control objetivo y localizar la propiedad llamada **ContextMenuStrip** (que actualmente tiene el valor **ninguno**) y mediante el desplegable, seleccionar el menú de contexto que le queremos asociar.

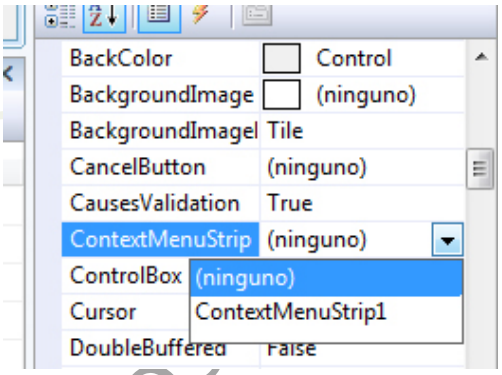


Figura: Asociando un menú de contexto al control

Asociamos nuestro menú de contexto al form1. Una vez hemos hecho esto, veremos que si ejecutamos la aplicación, ya podemos visualizar nuestro menú cuando utilizamos el botón secundario del ratón sobre cualquier punto del formulario.

Ahora añadimos a nuestro proyecto un control del tipo **PictureBox**. Cuando ya está hecho, volvemos a dejar la propiedad **ContextMenuStrip** del form1 a **ninguno** y a continuación, asociamos el menú de contexto al control de imagen que acabamos de añadir. Comprobamos que cuando ejecutamos la aplicación, solo se visualiza el menú de contexto si utilizamos el botón secundario del ratón con el puntero situado encima de la zona ocupada por la caja de la imagen.

Con esto podemos generar tantos menús de contexto como nos hagan falta y cada uno de ellos los podemos asociar a los controles correspondientes.

2.2.3. Barras de herramientas

Las barras de herramientas tienen una función similar a la barra de menús con la diferencia que al ser iconográficas son más intuitivas e independientes del idioma de la aplicación. En general, se utilizan como refuerzo o soporte a la barra de menú principal, y muy a menudo, duplicando alguna de las opciones más utilizadas.

Para añadir una barra de herramientas, hay que añadir un control del tipo **ToolStrip** a nuestro formulario. Entonces podemos ir añadiendo cada uno de los elementos a la barra. Para añadir un nuevo elemento hacemos clic en la flecha pequeña de color negro para desplegar el menú, y escogemos el que queramos:

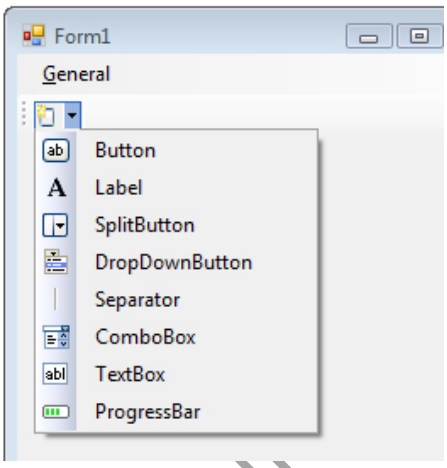


Figura: Añadir un nuevo elemento a una barra de herramientas

De momento añadiremos un elemento del tipo **Button**. De la misma forma que hemos hecho con la barra de menú, podemos editar la propiedad **Image** para escoger la imagen que queremos para el nuevo elemento, y si hacemos doble clic encima, el entorno nos genera el evento donde desarrollar el código asociado al elemento.

Ahora vamos a duplicar todo el menú principal en la barra de herramientas y la aplicación ha de quedar como muestra la siguiente imagen:

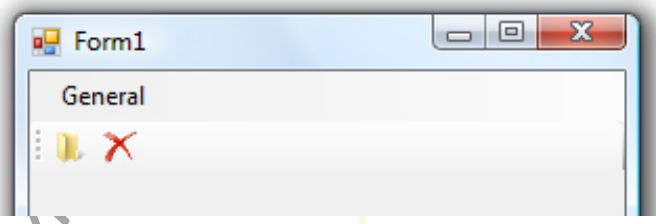


Figura: Barra de herramientas

A menudo, las barras de herramientas no son muy intuitivas y nos puede costar saber cuál es la funcionalidad de cada una de ellas. En la mayoría de las aplicaciones, si acercamos el puntero del ratón a un icono y lo dejamos unos instantes, aparece un rótulo flotando de color amarillento con un mensaje de ayuda.

Para añadir esta ayuda a nuestra aplicación, hemos de añadir un control del tipo **ToolTip** que lo podemos encontrar en la sección de **Controles Comunes** y añadirlo a nuestro proyecto. Este es uno de los controles que no se visualiza, sino que queda representado en la parte inferior de la zona de diseño. Una vez que hemos añadido este control, tenemos suficiente con editar la propiedad **ToolTipText** de cada uno de los elementos de la barra de herramientas.

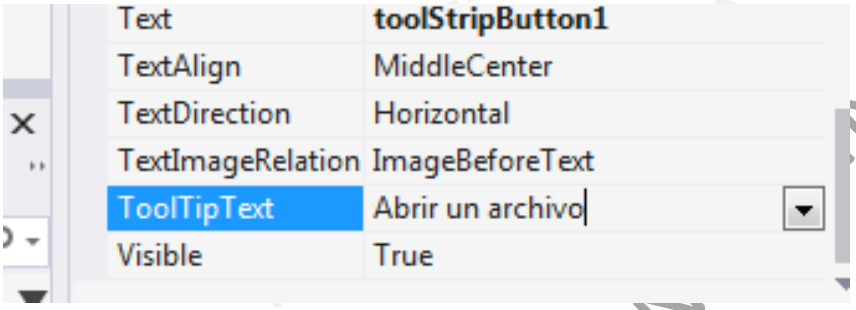


Figura: Texto de ayuda (I)

Si ejecutamos la aplicación, podemos comprobar que aparecen estos elementos de ayuda.

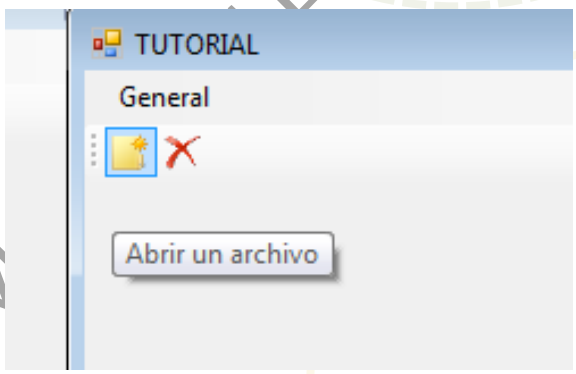


Figura: Texto de ayuda (II)

Con todo esto, cerramos el capítulo dedicado al diseño del interfaz del programa.

Conceptos clave

- **Mejoras:** En este tercer tema, hemos aprendido a mejorar el interfaz del programa. Hemos añadido menús, barras de herramientas así como cuadros de diálogo**.**
- **Múltiples Ventanas:** También hemos aprendido a gestionar aplicaciones con múltiples formularios.

Test de autoevaluación

1. Cual de las siguientes es la variable que contiene el resultado de un mensaje de alerta:
   a) Result.
   b) FormResult.
   c) AlertResult.
   d) DialogResult.

2. Cual de las siguientes es cierta para una ventana en forma modal::
   a) El programa se detiene hasta que cerramos la ventana.
   b) La ventana no se puede maximizar.
   c) Las ventanas modales solo se pueden usar una vez, luego hay que crearlas de nuevo.
   d) Las ventanas modales se pueden iniciar un dentro de la otra.

3. Cual de las siguientes afirmaciones es falsa:
   a) Los Menús sirven para ahorrar espacio en nuestra interfaz.
   b) Solo puede haber un menú en cada ventana.
   c) Los Menús permiten organizar las funcionalidades disponibles.
   d) Los Menús permiten crear atajos de teclado para los usuarios que no disponen de ratón.

4. Cual de los siguientes nombres creará un atajo Alt+a para la opción Salir de un menú:
   a) S%alir.
   b) S[a]lir.
   c) Sa&lir.
   d) S&alir.

5. Como se llama el control que permite crear una barra de herramientas en nuestra interfaz:
   a) StripBar.
   b) ToolBar.
   c) ToolStrip.
   d) ActionBar.

6. Cual de las siguientes propiedades permite mostrar un breve texto de ayuda en un icono:
   a) Hint.
   b) ToolTipText.
   c) HelpText.
   d) TipText.

Ponlo en práctica

Actividad 1

Vamos a mejorar nuestro programa de visionado de partidos de futbol desarrollado en el tema anterior. Para ello vamos a añadir los controles necesarios para visualizar los eventos que se van sucediendo en el partido. Vamos a visualizar 4 tipos de evento: Goles, tarjetas amarillas, tarjetas rojas y los inicios y fin de cada parte. Para cada evento debemos mostrar el tipo de evento, el minuto de juego y un texto descriptivo.

Debemos proporcionar los controles necesarios para gestionar la lista de eventos, así como añadir un menú principal y una barra de herramientas a nuestra interfaz.

Solucionarios

Test de autoevaluación

1. Cual de las siguientes es la variable que contiene el resultado de un mensaje de alerta:
   a) Result.
   b) FormResult.
   c) AlertResult.
   d) DialogResult.

2. Cual de las siguientes es cierta para una ventana en forma modal::
   a) El programa se detiene hasta que cerramos la ventana.
   b) La ventana no se puede maximizar.
   c) Las ventanas modales solo se pueden usar una vez, luego hay que crearlas de nuevo.
   d) Las ventanas modales se pueden iniciar un dentro de la otra.

3. Cual de las siguientes afirmaciones es falsa:
   a) Los Menús sirven para ahorrar espacio en nuestra interfaz.
   b) Solo puede haber un menú en cada ventana.
   c) Los Menús permiten organizar las funcionalidades disponibles.
   d) Los Menús permiten crear atajos de teclado para los usuarios que no disponen de ratón.

4. Cual de los siguientes nombres creará un atajo Alt+a para la opción Salir de un menú:
   a) S%alir.
   b) S[a]lir.
   c) Sa&lir.
   d) S&alir.

5. Como se llama el control que permite crear una barra de herramientas en nuestra interfaz:
   a) StripBar.
   b) ToolBar.
   c) ToolStrip.
   d) ActionBar.

6. Cual de las siguientes propiedades permite mostrar un breve texto de ayuda en un icono:
   a) Hint.
   b) ToolTipText.
   c) HelpText.
   d) TipText.

Ponlo en práctica

Actividad 1

Vamos a mejorar nuestro programa de visionado de partidos de futbol desarrollado en el tema anterior. Para ello vamos a añadir los controles necesarios para visualizar los eventos que se van sucediendo en el partido. Vamos a visualizar 4 tipos de evento: Goles, tarjetas amarillas, tarjetas rojas y los inicios y fin de cada parte. Para cada evento debemos mostrar el tipo de evento, el minuto de juego y un texto descriptivo.

Debemos proporcionar los controles necesarios para gestionar la lista de eventos, así como añadir un menú principal y una barra de herramientas a nuestra interfaz.

Solución:

Este ejercicio podrás descargarlo en la versión interactiva