

Trabajar con controles

Índice

Descripción.....	1
Lección: Crear un controlador de eventos para un control.....	2
Lección: Uso de los controles de los formularios Windows Forms	12
Lección: Uso de los cuadros de diálogo en una aplicación Windows Forms.	33
Lección: Agregar controles en tiempo de ejecución.....	43
Lección: Crear menús	49
Lección: Validar introducción de datos de los usuarios.....	59

Descripción

- Crear un controlador de eventos para un control
- Uso de los controles de los formularios Windows Forms
- Uso de los cuadros de diálogo en una aplicación Windows Forms
- Agregar controles en tiempo de ejecución
- Crear menús
- Validar introducción de datos de los usuarios

Introducción

Cuando diseñamos el interfaz de usuario (IU) y escribimos el código que opera detrás del IU de una aplicación, necesitamos trabajar con controles y sus eventos, propiedades y métodos para satisfacer los requerimientos de diseño.

Este módulo explica cómo crear *procedimientos* (o controladores) de eventos en nuestra aplicación que se ejecutarán en respuesta a acciones de los usuarios. Estudiaremos cómo añadir lógica de programación a los procedimientos de eventos de un control, cómo utilizar controles intrínsecos, cuadros de diálogo y menús de los formularios Windows Forms del Microsoft® .NET Framework, y cómo validar los datos introducidos por los usuarios de nuestra aplicación.

Objetivos

En este módulo, estudiaremos cómo:

- Crear un controlador de eventos para un control.
- Seleccionar y utilizar los controles adecuados en una aplicación Windows Forms.
- Utilizar cuadros de diálogo en una aplicación Windows Forms.
- Añadir controles a un formulario en tiempo de ejecución.
- Crear y utilizar menús en una aplicación Windows Forms.
- Validar entrada de datos del usuario en una aplicación Windows Forms.

Lección: Crear un controlador de eventos para un control

CONTENIDO

- Modelo de eventos del .NET Framework
- ¿Qué es un controlador de eventos?
- La palabra clave **Handles**
- Cómo crear controladores de eventos para eventos de control
- Cómo añadir y eliminar controladores de eventos en tiempo de ejecución
- Práctica: crear un controlador de eventos para un control

Introducción

En el Microsoft .NET Framework, un evento es un mensaje enviado por un objeto para indicar que se ha producido una acción invocada por un usuario o programáticamente. Cada evento tiene un emisor que produce el evento y un receptor que lo captura.

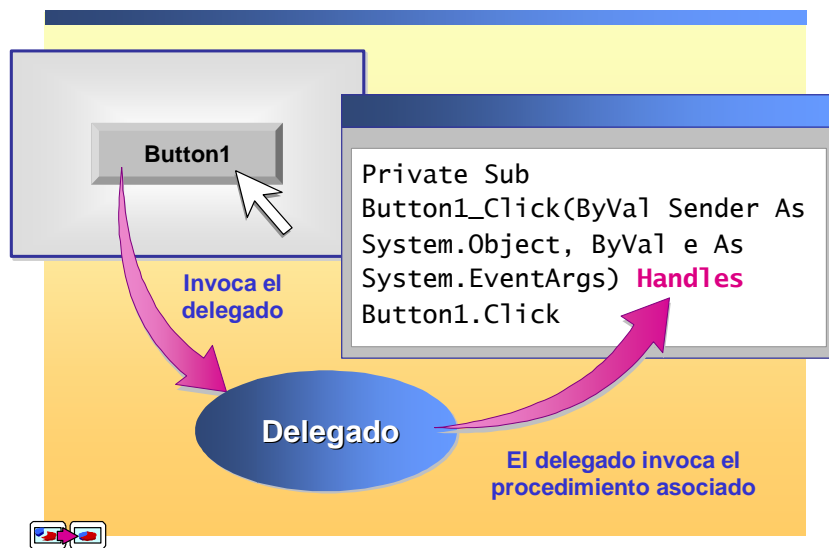
En esta lección, estudiaremos los eventos y los modos en que los eventos pueden ser controlados en nuestra aplicación. Estudiaremos cómo crear procedimientos que controlen los eventos y cómo añadir y eliminar controladores de eventos en tiempo de ejecución.

Objetivos de la lección

En esta lección, estudiaremos cómo:

- Describir el modelo de eventos del .NET Framework.
- Crear y utilizar controladores de eventos.
- Crear procedimientos de eventos utilizando las palabras clave **Handles** y **WithEvents**.
- Añadir y eliminar controladores de eventos desde procedimientos de eventos en tiempo de ejecución.

Modelo de eventos del .NET Framework



Introducción

En el .NET Framework, un evento se utiliza para indicar que se ha producido una acción. Por ejemplo, esta acción podría ser invocada por el usuario, como el evento **Click** de un control **Button**, o el evento podría producirse programáticamente para indicar el final de un largo cálculo.

Eventos y delegados

El objeto que produce (desencadena) el evento se denomina *emisor* del evento. El procedimiento que captura el evento se denomina *receptor* del evento. En cualquier caso, el emisor no sabe qué objeto o método responderá a los eventos que produzca. Por ello, es necesario tener un componente que enlace el emisor del evento con el receptor del evento. El .NET Framework utiliza un tipo de delegado para trabajar como un puntero a función entre el emisor y el receptor del evento. En la mayoría de casos, el .NET Framework crea el delegado y se ocupa de gestionar los detalles por nosotros. Sin embargo, podemos crear nuestros propios delegados para los casos en que deseemos que un evento utilizar diferentes controladores de eventos en diferentes circunstancias.

Crear delegados

Los delegados son objetos que podemos utilizar para invocar métodos de otros objetos. Podemos utilizar la palabra clave **Delegate** en una sentencia de declaración para crear nuestro propio delegado que derive de la clase **MulticastDelegate**. Crear nuestros propios delegados puede ser útil en situaciones en las que necesitamos un objeto intermediario entre un procedimiento que emite la llamada y el procedimiento al que se llama. Si deseamos más información sobre la creación y uso de delegados, realizar una búsqueda utilizando la frase **Delegate Class** en la documentación de ayuda de Microsoft Visual Studio® .NET.

Palabra clave Handles

El .NET Framework también proporciona la palabra clave **Handles** como una forma sencilla de asociar un procedimiento de eventos, o controlador, a un evento. La palabra clave **Handles** asocia un procedimiento a un evento que ha sido producido por un objeto declarado utilizando la palabra clave **WithEvents**. Debido a que cada control que añadimos a un formulario se declara automáticamente utilizando la palabra clave **WithEvents**, normalmente asociaremos el evento de un control a un procedimiento de eventos utilizando **Handles**.

¿Qué es un controlador de eventos?

■ Controladores de eventos

- Métodos ligados a un evento
- Cuando el evento es raised, se ejecuta el código del controlador de eventos

■ Dos argumentos de eventos con controladores de eventos

- Un objeto que representa el objeto que raised el evento
- Un objeto de evento que contiene cualquier información específica del evento

```
Private Sub Button1_Click (ByVal Sender As
System.Object, ByVal e As System.EventArgs)
```

Introducción

Añadimos funcionalidad a los controles produciendo y consumiendo eventos. Antes de que nuestra aplicación pueda responder a un evento, debemos crear un controlador de eventos. El controlador de eventos (procedimiento de eventos) contiene la lógica de programa que se ejecuta cuando se produce el evento.

Definición

Un controlador de eventos es un método (normalmente, un procedimiento *Sub*) ligado a un evento. Cuando se produce el evento, se ejecuta el código del controlador de eventos. Podemos utilizar un mismo controlador de eventos para controlar más de un evento. Por ejemplo, podemos crear un solo controlador de eventos que controle los eventos de un botón y un elemento de menú que se utilicen para lo mismo. Igualmente, si tenemos un grupo de controles **RadioButton** en un formulario, podríamos crear un solo controlador de eventos y que el evento **Click** de cada control estuviese ligado al controlador de eventos.

Importante Microsoft Visual Basic® .NET ya no soporta matrices de control. Los cambios realizados al modelo de eventos hacen innecesarias las matrices de controles. Del mismo modo en que las matrices de controles en Visual Basic 6.0 podían compartir eventos, el modelo de eventos en Visual Basic .NET permite que cualquier controlador de eventos controle eventos desde múltiples controles. De hecho, esto nos permite crear grupos de controles de diferentes tipos que comparten los mismos eventos.

Ejemplo de controlador de eventos

El siguiente ejemplo de código es un controlador de eventos para el evento **Click** de un botón.

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

El siguiente código de ejemplo muestra cómo podemos utilizar un solo controlador de eventos para controlar eventos para múltiples controles.

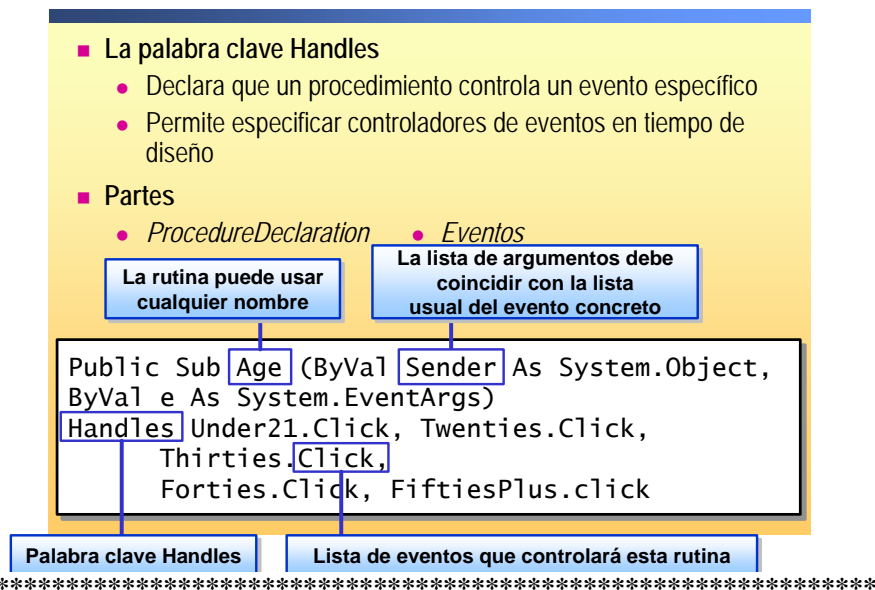
```
Private Sub MyHandler(ByVal Sender As System.Object, ByVal e As System.EventArgs) Handles RadioButton1.Click, RadioButton2.Click, RadioButton3.Click
```

Parámetros del controlador de eventos

Cada controlador de eventos proporciona dos parámetros que permiten controlar el evento correctamente:

- El primer parámetro (*Sender* en el ejemplo de código anterior) proporciona una referencia al objeto que ha producido el evento. Especifica la fuente que ha producido el evento.
- El segundo parámetro (*e* en el ejemplo de código anterior) pasa un objeto específico al evento que se está controlando. Este parámetro contiene todos los datos necesarios para controlar el evento.

La palabra clave Handles



Introducción

El modo en que construimos un controlador de eventos depende del modo en que deseamos asociarlo a eventos. La manera estándar de crear controladores de eventos es utilizar la palabra clave **Handles** para objetos que se han declarado mediante la palabra clave **WithEvents**.

La palabra clave Handles

La palabra clave **Handles** permite declarar controladores de eventos en tiempo de diseño. Se utiliza para declarar que un procedimiento controla un evento específico. Utilizar la palabra clave **Handles** al final de una declaración de procedimiento para conseguir que controle eventos producidos por una variable de objeto declarada utilizando la palabra clave **WithEvents**. La palabra clave **Handles** también puede utilizarse en una clase derivada para controlar eventos de una clase base.

Partes de la palabra clave Handles

La palabra clave **Handles** utiliza la siguiente declaración:

```
Proceduredeclaration Handles event
```

■ *Proceduredeclaration*

Proceduredeclaration es la declaración del procedimiento Sub del procedimiento que controlará el evento.

■ *event*

event es el nombre del evento que se está controlando. Este evento debe ser generado por la clase base de la clase actual o por un objeto declarado mediante la palabra clave **WithEvents**.

Cómo crear controladores de eventos para eventos de control

- Utilizar la palabra clave **WithEvents** para declarar variables de objetos que se utilizarán con la instrucción **Handles**
- Utilizar la palabra clave **Handles** al final de la declaración del procedimiento

```
Friend WithEvents Button1 As System.Windows.Forms.Button  
  
Private Sub Button1_Click(ByVal sender As System.Object,  
    ByVal e As System.EventArgs) Handles Button1.Click  
  
    MessageBox.Show("MyHandler captured the event")  
  
End Sub
```

Introducción

La forma estándar de crear controladores (procedimientos de eventos) en Visual Basic .NET es utilizar la palabra clave **Handles**. La palabra clave **Handles** trabaja con objetos declarados mediante la palabra clave **WithEvents**. Debido a que los controles que añadimos a un formulario se declaran automáticamente utilizando la palabra clave **WithEvents**, el siguiente procedimiento es aplicable a los eventos producidos por controles.

Procedimiento

Para crear un procedimiento de eventos que utilizar la palabra clave **Handles**:

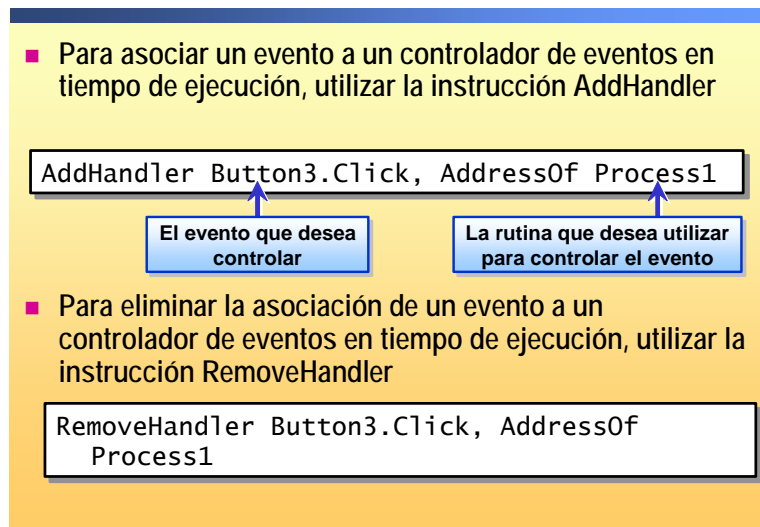
1. En la sección de declaraciones del módulo que controlará el evento, utilizar la palabra clave **WithEvents** para declarar una variable de objeto para la fuente de sus eventos. Para un control añadido a un formulario, esto se realiza automáticamente por nosotros, como en el siguiente código de ejemplo:

```
Friend WithEvents Button1 As System.Windows.Forms.Button
```

2. En el Editor de código, en la lista **Nombre de clase**, hacer clic en la variable de objeto que acabamos de declarar. Éste es el objeto que se ha declarado utilizando la palabra clave **WithEvents**.
3. En la lista **Nombre de método**, hacer clic en el evento que deseamos controlar. El Editor de código crea el procedimiento del controlador de eventos vacío con una cláusula **Handles**.
4. Añadir código de control de eventos al procedimiento del controlador de eventos utilizando los argumentos que se proporcionan. El siguiente código proporciona un ejemplo:

```
Sub Button_Click(ByVal sender As Object, ByVal e As  
    System.EventArgs) Handles Button1.Click  
  
    MessageBox.Show("MyHandler received the event")  
  
End Sub
```

Cómo añadir y eliminar controladores de eventos en tiempo de ejecución



Introducción

En Visual Basic .NET, podemos añadir y eliminar controladores de eventos en tiempo de ejecución utilizando las instrucciones **AddHandler** y **RemoveHandler**. La instrucción **AddHandler** es similar a la cláusula **Handles**; ambos permiten especificar un controlador de eventos que controlará un evento. Sin embargo, **AddHandler** junto con **RemoveHandler** proporcionan una mayor flexibilidad que la cláusula **Handles** y permiten añadir, eliminar y cambiar dinámicamente el controlador de eventos asociado a un evento. A diferencia de la palabra clave **Handles**, **AddHandler** permite asociar múltiples controladores de eventos a un único evento.

Procedimiento: añadir controladores de eventos utilizando AddHandler

Para añadir controladores de eventos utilizando **AddHandler**:

1. Declarar una variable de objeto de la clase que sea la fuente de los eventos que deseamos controlar. Por ejemplo:

```
Dim Button1 As System.Windows.Forms.Button
```

2. Utilizar la instrucción **AddHandler** para especificar el nombre del emisor del evento, y la instrucción **AddressOf** para proporcionar el nombre de su controlador de eventos, como se muestra en el siguiente código de ejemplo:

```
AddHandler Button3.Click, AddressOf Process1
```

3. Añadir código al controlador de eventos, como en el siguiente código de ejemplo:

```
Private Sub Process1(ByVal sender As System.Object, _
                    ByVal e As System.EventArgs)
    MessageBox.Show("Use Process 1 to Perform the Task")
End Sub
```

Procedimiento: eliminar controladores de eventos utilizando **RemoveHandler**

Para eliminar controladores de eventos utilizando **RemoveHandler**:

- Utilizar la instrucción **RemoveHandler** para especificar el nombre del emisor del evento, y la instrucción **AddressOf** para proporcionar el nombre de su controlador de eventos.

```
RemoveHandler Button3.Click, AddressOf Process1
```

Práctica: crear un controlador de eventos para un control



En esta práctica,

- Crearemos un controlador de eventos para un evento **MouseMove**
- Crearemos un controlador de eventos para un evento **Click**

Empezar revisando los objetivos de esta actividad práctica

5 min 

Introducción

En esta práctica, crearemos un controlador de eventos para un control.

Instrucciones

🔍 Abrir el proyecto de la práctica

1. Utilizando Windows Explorer, vaya a la carpeta pract01\Starter. Esta carpeta se puede encontrar dentro del fichero practs07.zip. La solución del ejercicio práctico se puede encontrar en la carpeta pract01\Solution dentro del mismo fichero comprimido.
2. Hacer doble clic en el archivo de solución eventHandlers.sln para abrir el proyecto.

🔍 Crear un controlador de eventos para un evento MouseMove

1. Abrir Form1.vb en el Editor de código.
2. En la lista **Nombre de clase**, hacer clic en **csButton**.
3. En la lista **Nombre de método**, hacer clic en **MouseMove**.
4. Añadir las siguientes instrucciones de código a la subrutina **csButton_MouseMove**:

```
csButton.Top -= e.Y
csButton.Left += e.X
If csButton.Top < -16 Or csButton.Top > 160 Then _
csButton.Top = 73
If csButton.Left < -64 Or csButton.Left > 384 Then _
csButton.Top = 160
```

5. ¿Cuál es el objetivo del segundo parámetro (e) que se pasa a este controlador de eventos?

El parámetro e contiene los datos del evento. Es un objeto EventArgs (la clase base que no contiene datos del evento) o es una instancia de una clase derivada como MouseEventArgs. Para ver una lista completa de las clases derivadas, realizar una búsqueda utilizando la frase EventArgs Class en la documentación de Ayuda de Visual Studio .NET y el enlace siguiente a 'Clases derivadas'.

6. Ejecutar la aplicación y hacer clic en cada botón del formulario.
7. Cerrar la aplicación.

⏏ Crear un controlador de eventos para un evento Click

1. Abrir Form1.vb en la vista Diseño.
2. Hacer doble clic en el control del botón **Close**.
3. ¿Por qué se crea un controlador de eventos **Click**?

El IDE crea automáticamente un controlador para el evento predeterminado cuando hacemos doble clic en un control en la vista de Diseño.

4. Añadir la siguiente instrucción de código a la subrutina **closeButton_Click**:
End
5. Ejecutar la aplicación y hacer clic en el botón **Close**.
6. Guardar el proyecto y cerrar Visual Studio.

Lección: utilizar los controles de los formularios Windows Forms

- Seleccionar un control de un formulario Windows Forms basándose en la función
- Cómo utilizar el control **StatusBar**
- Cómo utilizar el control **ListBox**
- Cómo utilizar los controles **GroupBox** y **Panel**
- Cómo utilizar los controles **ToolBar** e **ImageList**
- Práctica: crear y utilizar un control **ToolBar**
- Demostración: implementar operaciones de arrastrar y soltar entre controles

Introducción

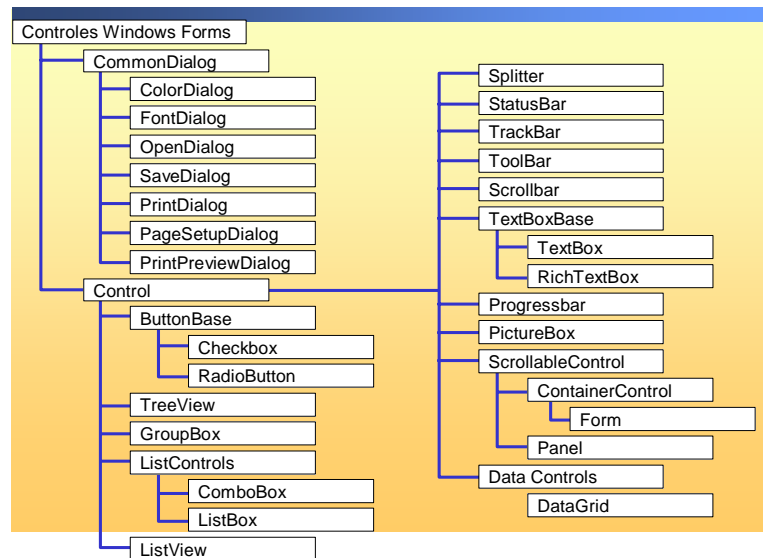
La pestaña **Windows Forms** del Cuadro de herramientas de Visual Studio .NET ofrece varios controles útiles, como los controles **ToolBar**, **StatusBar**, **ListBox**, **GroupBox**, **ImageList**, **OpenFileDialog** y **MainMenu**. Los controles del Cuadro de herramientas pueden clasificarse en base a la funcionalidad que añaden a las aplicaciones. En esta lección, estudiaremos las diferentes categorías de controles de los formularios Windows Forms y cómo utilizar controles desde las categorías. Algunos de los controles se explicarán más detalladamente en las lecciones *Uso de los cuadros de diálogo en una aplicación Windows Forms* y *Creación de menús* de este módulo.

Objetivos de la lección

En esta lección, estudiaremos cómo:

- Seleccionar los controles Windows Forms adecuados para añadir la funcionalidad deseada a una interfaz de usuario.
- Utilizar el control **StatusBar** en una aplicación Windows Forms para mostrar información en texto.
- Utilizar el control **ListBox** en una aplicación Windows Forms para proporcionar al usuario una lista predefinida de elementos.
- Utilizar los controles **GroupBox** y **Panel** en una aplicación Windows Forms como contenedores de otros controles.
- Utilizar los controles **ToolBar** e **ImageList** en una aplicación Windows Forms para mostrar botones de comandos como una matriz de imágenes gráficas.
- Implementar operaciones de arrastrar y soltar entre controles.

Seleccionar un control Windows Forms según la función



Introducción

Los controles Windows Forms son componentes reutilizables que encapsulan la funcionalidad de la interfaz de usuario y se utilizan en una aplicación Windows Forms. La biblioteca de clases del .NET Framework no sólo proporciona numerosos controles listos para ser usados, sino que proporciona además la infraestructura para desarrollar nuestros propios controles.

Los controles que aparecen en la pestaña Windows Forms del Cuadro de herramientas pueden clasificarse en base a sus funciones. Dependiendo de la funcionalidad que deseemos proporcionar en la interfaz de usuario de nuestra aplicación, seleccionaremos un control de alguna de las siguientes categorías: Comandos, Texto, Opciones, Contenedores, Gráficos, Menús o Cuadros de diálogo. Observar que la jerarquía de clases y la categoría funcional de un control no siempre coinciden.

Controles de la categoría comandos

Las siguientes son categorías de controles de comandos:

■ **Button**

Se utiliza para iniciar, detener o interrumpir un proceso.

■ **LinkLabel**

Muestra texto en forma de vínculo en estilo Web y desencadena un evento cuando el usuario hace clic en el texto especial. Normalmente, el texto es un vínculo a otra ventana o a un sitio Web.

■ **NotifyIcon**

Muestra un icono, en el área de notificación del estado en la barra de tareas, que representa una aplicación que se ejecuta en segundo plano.

■ **ToolBar**

Contiene una colección de controles de botones.

Controles de la categoría texto

Los siguientes controles de texto se utilizan para permitir a los usuarios introducir y editar el texto que contienen estos controles en tiempo de ejecución:

- **TextBox**

Muestra texto escrito en tiempo de diseño que puede ser editado por los usuarios en tiempo de ejecución o modificado mediante programación.

- **RichTextBox**

Habilita la presentación del texto con formato de texto sencillo o de texto enriquecido (RTF).

Los siguientes controles de texto adicionales pueden utilizarse para mostrar texto pero no permiten a los usuarios de la aplicación editar directamente el contenido de texto que muestran:

- **Label**

Muestra texto que los usuarios no pueden editar directamente.

- **StatusBar**

Muestra información acerca del estado actual de la aplicación mediante una ventana con marco. Normalmente, la barra de estado se encuentra en la parte inferior de un formulario primario.

Controles de la categoría opciones

La siguiente selección de controles permite a los usuarios seleccionar un valor de una lista:

- **CheckedListBox**

Muestra una lista desplazable de elementos, cada uno acompañado por una casilla de verificación.

- **ComboBox**

Muestra una lista desplegable de elementos.

- **DomainUpDown**

Muestra una lista de elementos de texto por la cual los usuarios se pueden desplazar mediante botones hacia arriba y hacia abajo.

- **ListBox**

Muestra una lista de texto y elementos gráficos (iconos).

- **ListView**

Muestra los elementos en una de cuatro vistas diferentes. Las vistas incluyen Sólo texto, Texto con iconos pequeños, Texto con iconos grandes y una vista de Detalles.

- **NumericUpDown**

Muestra una lista de números a través de la cual los usuarios se pueden desplazar mediante botones hacia arriba y hacia abajo.

- **TreeView**

Muestra una colección jerárquica de objetos de nodo que pueden estar formados por texto con casillas de verificación o iconos opcionales.

Controles de la categoría contenedor

Los controles de contenedor pueden utilizarse para agrupar otros controles en un formulario. Algunos ejemplos de controles de contenedor son:

- **Panel**

Agrupar un conjunto de controles en un marco sin etiqueta que permite el desplazamiento.

- **GroupBox**

Agrupar un conjunto de controles (como botones de opciones) en un marco con etiqueta que no permite desplazamiento.

- **TabControl**

Proporciona una página con pestañas para organizar y tener acceso a controles agrupados eficazmente.

Controles de la categoría gráficos

Las siguientes son Controles de la categoría de gráficos:

- **ImageList**

Sirve como repositorio de imágenes. Los controles ImageList y las imágenes que contienen pueden reutilizarse de una aplicación a la siguiente.

- **PictureBox**

Muestra archivos gráficos, como mapas de bits e iconos, en un marco.

Controles de la categoría cuadros de diálogo

Visual Studio .NET proporciona una serie de cuadros de diálogo comunes, entre los cuales se incluyen **ColorDialog**, **FontDialog**, **PageSetupDialog**, **PrintDialog**, **OpenFileDialog**, etc. Estudiaremos más acerca de los cuadros de diálogo en la lección *Uso de los cuadros de diálogo en una aplicación Windows Forms*, en este módulo.

Controles de la categoría menú

Las siguientes son Controles de la categoría menú:

- **MainMenu**

Proporciona una interfaz en tiempo de diseño para la creación de menús.

- **ContextMenu**

Implementa un menú que aparece cuando el usuario hace clic en un objeto con el botón secundario del ratón.

Cómo utilizar el control StatusBar

1	Añadir un control StatusBar al formulario
2	Hacer clic en la propiedad Panels y abrir el Editor de colecciones StatusBarPanel
3	Utilizar los botones Agregar y Quitar para añadir y eliminar paneles del control StatusBar
4	Configurar las propiedades de los paneles individuales
5	Hacer clic en Aceptar para cerrar el cuadro de diálogo y crear los paneles que se han especificado
6	En la ventana Propiedades, establecer la propiedad ShowPanels como true

Introducción

El control **StatusBar** es un ejemplo interesante de control utilizado para mostrar información textual. Una barra de estado es una ventana horizontal en la parte inferior de una ventana primaria en la que una aplicación puede mostrar distintos tipos de información de estado. La barra de estado puede dividirse en partes para mostrar más de un tipo de información. Los controles **StatusBar** pueden tener paneles de barra de estado que muestren texto o iconos indicando el estado, o una serie de iconos en una animación que indiquen que un proceso está funcionando, como la barra de estado de Microsoft Word que indica que un documento se está guardando.

Procedimiento: Uso del control StatusBar

El .NET Framework ofrece el control **StatusBar** para la barra de estado. Podemos crear paneles en la barra de estado utilizando el método **Add** de la colección **Panels**. Para mostrar los paneles, debemos establecer la propiedad **ShowPanels** como **True**. Podemos indicar el tamaño y alineación de cada panel estableciendo propiedades adicionales.

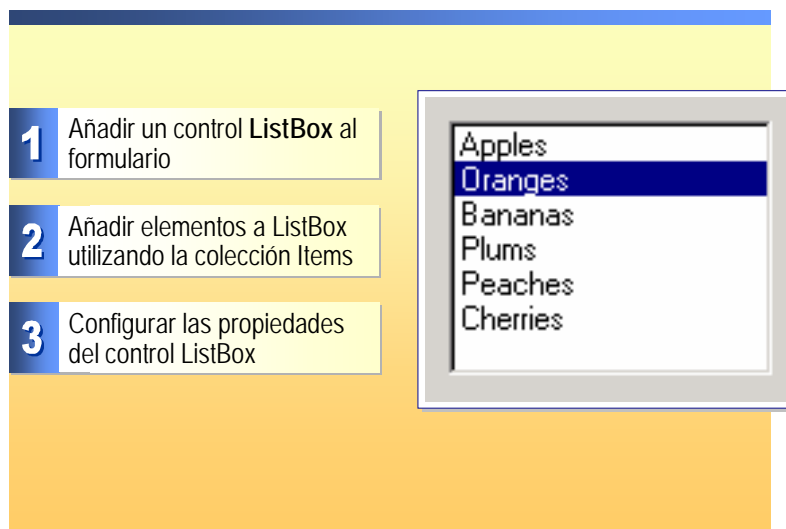
Para crear una barra de estado con paneles:

1. Añadir un control **StatusBar** al formulario.
2. En la ventana Propiedades, hacer clic en la propiedad **Panels** para seleccionarla. A continuación, en el botón con puntos suspensivos (...) para abrir el Editor de colecciones StatusBarPanel.
3. Utilizar los botones **Agregar** y **Quitar** para añadir y eliminar paneles del control **StatusBar** en tiempo de diseño. Podemos utilizar los métodos **Add** y **Remove** del objeto **StatusBarPanels** para añadir y eliminar paneles en tiempo de ejecución.
4. Configurar las propiedades de los paneles individuales en la ventana Propiedades. La siguiente tabla muestra una lista de las propiedades importantes y su descripción:

Propiedad	Descripción
AutoSize	Determina el comportamiento del panel en ajuste de tamaño.
Alignment	Establece la alineación del panel en el control StatusBar .
BorderStyle	El estilo del borde del panel que se muestra.
Icon	El icono (archivo .ico) que se muestra en el panel.
Style	Establece el estilo del panel. Debe ser uno de los valores de la enumeración StatusBarPanelStyle .
Text	La cadena de texto que se muestra en el panel.
MinWidth	El ancho mínimo del panel en la barra de estado.

5. Hacer clic en **Aceptar** para cerrar el cuadro de diálogo y crear los paneles que hemos especificado.
6. En la ventana Propiedades, establecer la propiedad **ShowPanels** como **True**.

Cómo utilizar el control ListBox



Introducción

El control **ListBox** es un buen ejemplo de la categoría de opciones. El control **ListBox** de Windows Forms muestra una lista de elementos de los cuales el usuario puede seleccionar uno o más. Si el número total de elementos supera el número que puede mostrarse, automáticamente se agrega una barra de desplazamiento al control **ListBox**.

Propiedades de ListBox

La siguiente tabla muestra una lista de las propiedades importantes del control **ListBox**:

Propiedad	Descripción
MultiColumn	Cuando se establece en True , el cuadro de lista muestra elementos en múltiples columnas y aparece una barra de desplazamiento horizontal. Cuando se establece en False , el cuadro de lista muestra elementos en una sola columna y aparece una barra de desplazamiento vertical.
ScrollAlwaysVisible	Cuando se establece en True , la barra de desplazamiento aparece independientemente del número de elementos.
SelectionMode	Determina cómo pueden seleccionarse a la vez varios elementos de la lista.
SelectedIndex	Devuelve un valor entero que corresponde al primer elemento seleccionado de la lista. Si no se selecciona ningún elemento, el valor de SelectedIndex es -1 . Si se selecciona el primer elemento de la lista, el valor de SelectedIndex es 0 .
Items.Count	Refleja el número de elementos de la lista.
Items.Add/Items.Insert	Agrega elementos al control ListBox .
Items.Delete/Items.Clear	Elimina elementos del control ListBox .
DataSource	Vincula ListBox a una fuente de datos.
DisplayMember	Vincula ListBox a un nombre de columna en la fuente de datos.

Procedimiento: uso del control ListBox

Para utilizar un control **ListBox**:

1. Añadir un control **ListBox** al formulario.
2. Añadir elementos a **ListBox** utilizando la colección **Items**.

Podemos añadir múltiples elementos a **ListBox** al mismo tiempo utilizando el método **AddRange**.

```
ListBox1.Items.AddRange(NewObject() {"Apples", "Oranges",  
"Bananas"})
```

3. Configurar las propiedades del control **ListBox**.

Cómo utilizar los controles GroupBox y Panel

1	Arrastrar un control contenedor (Panel o GroupBox) de la Caja de herramientas a un formulario
2	Agregar otros controles al control contenedor, arrastrando cada uno al panel
3	Si se dispone de controles existentes que se desean encerrar en el contenedor, se deben arrastrar a éste
4	Para mostrar barras de desplazamiento para el control Panel , establecer su propiedad AutoScrollbar en True
5	Para mostrar una leyenda en el GroupBox, establecer su propiedad Text con la leyenda adecuada

Introducción

Cuando deseamos que el usuario seleccione una o más opciones de un grupo con varias opciones disponibles, normalmente utilizamos casillas de verificación (más de una selección) o botones de opción (una sola selección) agrupados en un control contenedor. Todos los controles de un control contenedor funcionan como un grupo. Las tres principales razones para agrupar controles son las siguientes:

- Agrupar visualmente elementos relacionados con el formulario para tener una interfaz de usuario clara
- Mover los controles como una unidad en tiempo de diseño
- Agrupación programática de controles

Visual Studio .NET incluye controles contenedor como **GroupBox** y **Panel** que permiten agrupar botones de opciones, casillas de verificación u otros controles que deseemos tratar como parte de una colección de controles. El control **Panel** es similar al control **GroupBox**, aunque el control **Panel** puede tener barras de desplazamiento, y únicamente el control **GroupBox** muestra un título.

Procedimiento: crear y poblar controles contenedor

Para crear y poblar un control contenedor:

1. Arrastrar un control contenedor (**Panel** o **GroupBox**) desde la etiqueta Windows Forms del Cuadro de herramientas a un formulario.
2. Añadir otros controles al control contenedor, arrastrando cada uno al interior del panel.
3. Si hay controles existentes que deseamos incluir en el contenedor, arrastrarlos hasta él.
4. Para mostrar barras de desplazamiento para el control **Panel**, establecer su propiedad **AutoScrollbar** en **True**.
5. Para mostrar un título en el **GroupBox**, establecer su propiedad **Text** con un título adecuado.

Puede accederse a los controles agrupados en un control contenedor utilizando la propiedad **Controls**. Cada control agrupado dentro de un **Panel** o un **GroupBox** es miembro del objeto **Control.ControlCollection**, que está asignado a la propiedad **Control** del contenedor. Estudiaremos con mayor detalle **ControlCollection** en la lección *Agregar controles en tiempo de ejecución*, de este módulo.

Cómo utilizar los controles **ToolBar** e **ImageList**

Para utilizar ToolBar en un formulario Windows Forms	
1	Añadir un control ToolBar desde la Caja de herramientas al formulario
2	Añadir botones al ToolBar
3	Añadir los botones al ToolBarButtonCollection
4	Configurar los botones estableciendo el texto y/o imagen

Introducción

El control **ToolBar** es un buen ejemplo de control utilizado para aceptar comandos del usuario. Las barras de herramientas son un elemento alternativo de interfaz gráfica de usuario (GUI) para los menús. Una barra de herramientas contiene una serie de botones representados por la clase **ToolBarButton** en el .NET Framework.

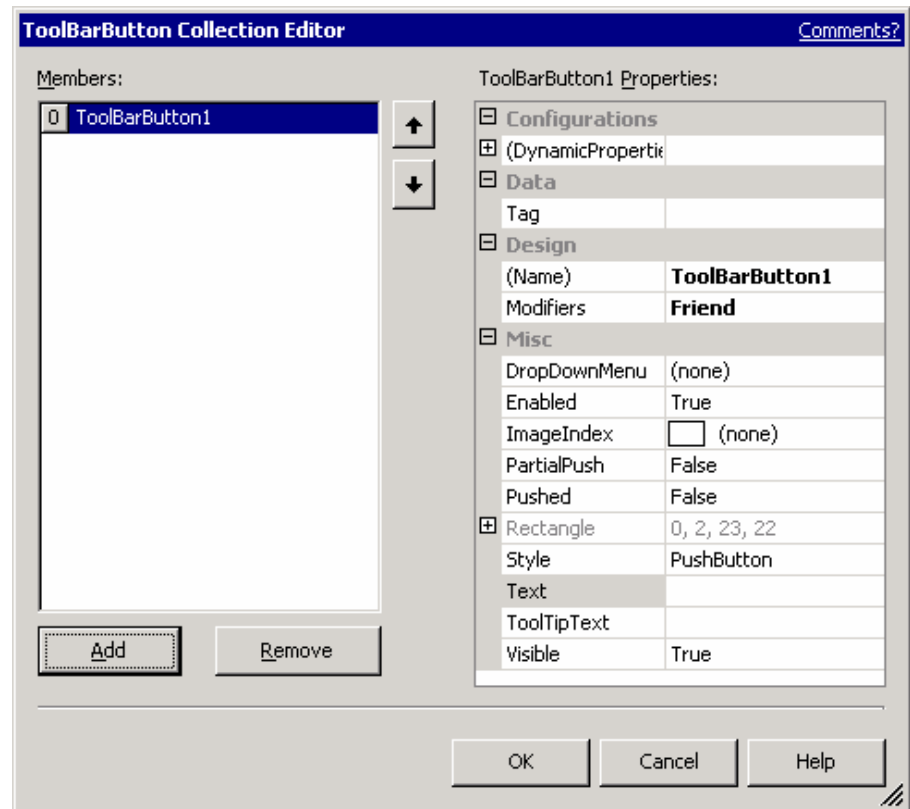
Procedimiento: uso de ToolBar

Para utilizar una barra de herramientas **ToolBar** en una aplicación Windows Forms:

1. Añadir un control **ToolBar** desde el Cuadro de herramientas al formulario.
2. Añadir botones al control **ToolBar**.

Podemos utilizar la propiedad **Buttons** para añadir botones al control **ToolBar**.

3. Añadir los botones al Editor de colecciones **ToolBarButton** utilizando la propiedad **Buttons**.



4. Configurar los botones estableciendo el texto, imagen, etc.

Cada botón de la barra de herramientas **ToolBar** puede tener su propio aspecto. El título y la imagen son opcionales. De modo similar al control **ListView** y **TreeView**, las imágenes de las barras de herramientas se almacenan en una lista de imágenes. La clase **ToolBarButton** tiene una propiedad **Index** que hace referencia a la imagen de la lista.

Los botones de la barra de herramientas pueden aparecer en distintos tipos. La propiedad **Style** puede establecer para uno de los valores de **ToolBarButtonStyle** que se muestran en la siguiente tabla:

Valor	Descripción
DropDownButton	Un control desplegable que muestra un menú u otra ventana cuando se hace clic en él.
PushButton	Un botón estándar en tres dimensiones.
Separator	Un espacio o línea entre botones de la barra de herramientas. El aspecto depende del valor de la propiedad Appearance .

ToggleButton	Un botón <i>toggle</i> que aparece hundido cuando se hace clic en él y conserva el aspecto hundido hasta que se hace clic en él de nuevo.
--------------	---

Nota Para mostrar información sobre un botón, establecer la propiedad **ShowToolTips** en **True**. Podemos definir el contenido de esta información estableciendo la propiedad **ToolTipText** del botón.

Procedimiento: Uso de ImageList

El control **ImageList** de los formularios Windows Forms se utiliza para almacenar imágenes, que pueden ser mostradas por controles. Por ejemplo, podemos habilitar el botón para mostrar diferentes imágenes modificando la propiedad **ImageIndex**. También podemos asociar la misma lista de imágenes a múltiples controles. Podemos utilizar una lista de imágenes a cualquier control que tenga una propiedad **ImageList** o, en el caso del control **ListView**, las propiedades **SmallImageList** y **LargeImageList**. Los controles que pueden asociarse a una lista de imágenes incluyen **ListView**, **TreeView**, **ToolBar**, **TabControl**, **Button**, **CheckBox**, **RadioButton** y **Label**.

Para asociar la lista de imágenes a un control, establecer la propiedad **ImageList** del control con el nombre del control **ImageList**. La propiedad clave del control **ImageList** es **Images**, que contiene las imágenes que utilizará el control asociado. Puede accederse a cada imagen individual por el valor de su índice. La propiedad **ColorDepth** determina el número de colores que de las imágenes. Todas las imágenes se muestran en el mismo tamaño, que está determinado por la propiedad **ImageSize**. Las imágenes que sean de mayor tamaño, se ajustarán.

Procedimiento:
desencadenar eventos
para botones de la Barra
de herramientas

Si nuestra aplicación Windows Forms tiene un control **ToolBar** con botones, seguramente desearemos saber qué botón pulsa el usuario. Para determinar qué botón se pulsa, se añade un controlador de eventos el evento **ButtonClick** del control **ToolBar**. Utilizar una instrucción **Select Case** y la clase **ToolBarButtonClickEventArgs** para determinar qué botón de la barra de herramientas se pulsa. El siguiente ejemplo muestra cómo utilizar la propiedad **Button** del objeto **ToolBarButtonClickEventArgs** para determinar qué botón se pulsa.

Nota El siguiente código de ejemplo utiliza la propiedad **Tag** para determinar qué control se pulsa, pero también se puede utilizar el valor del índice de un control. No obstante, el uso del valor del índice de los controles dificulta el seguimiento de los controles y sus valores de índice correspondientes. Por ejemplo, si tenemos un separador en nuestro formulario, el separador también utilizará un valor de índice, y necesitaremos tener en cuenta el separador cuando hacemos referencia al valor del índice.

```
Private Sub ToolBar1_ButtonClick(ByVal sender As _
System.Object, ByVal e As _
System.Windows.Forms.ToolBarButtonClickEventArgs) Handles _
ToolBar1.ButtonClick
    Select Case e.Button.Tag
        Case "Cut"
            Me.cutRadioButton.Checked = True
            Me.StatusBar1.Panels(0).Text = _
            Me.cutRadioButton.Text & _
            " Radio Button is checked"

        Case "Copy"
            Me.copyRadioButton.Checked = True
            Me.StatusBar1.Panels(0).Text = _
            Me.copyRadioButton.Text & _
            " Radio Button is checked"

    End Select

    MessageBox.Show("The " & e.Button.Tag & _
        " button is index number " & _
        ToolBar1.Buttons.IndexOf(e.Button) & " on the toolbar")
End Sub
```

Práctica: crear y utilizar un control ToolBar



En esta práctica,

- Añadiremos un control **ToolBar** y un control **ImageList**
- Añadiremos botones a un control **ToolBar**
- Añadiremos imágenes a un control **ToolBar**
- Asignaremos valores a las propiedades **Tag** y **ToolTipText** de los botones de **ToolBar**
- Crearemos un controlador de eventos para el evento **ButtonClick**

Empezar examinando los objetivos de esta actividad práctica



15 min

Introducción

En esta práctica, crearemos y utilizaremos un control **ToolBar**.

Instrucciones

🔍 Abrir el proyecto de la práctica

5. Utilizando Windows Explorer, abrir la carpeta pract02\Starter. Esta carpeta se puede encontrar dentro del fichero practs07.zip. La solución del ejercicio práctico se puede encontrar en la carpeta pract02\Solution dentro del mismo fichero comprimido.
6. Hacer doble clic en el archivo de solución **ToolBar.sln** para abrir el proyecto.

🔍 Añadir un control ToolBar y un control ImageList al proyecto

1. Abrir Form1.vb en vista de Diseño.
2. En el cuadro de herramientas, hacer doble clic en **ImageList**.
3. En el cuadro de herramientas, hacer doble clic en **ToolBar**.

🔍 Añadir botones al control ToolBar

1. En the formulario, hacer clic en **ToolBar1**.
2. En la ventana Propiedades de ToolBar1, hacer clic en **Buttons**.
3. En la columna de valores de la propiedad **Buttons**, hacer clic en el botón de **puntos suspensivos (...)**.
4. En la ventana del Editor de la colecciones ToolBarButton, hacer clic en **Agregar**.
5. Utilizar el botón **Agregar** para añadir seis botones más al control **ToolBar**.
6. Hacer clic en **ToolBarButton5** y, a continuación, hacer clic en **Style** en la tabla de propiedades.
7. En la lista de valores de la propiedad **Style**, hacer clic en **Separator**.

8. ¿Por qué no podemos añadir todavía imágenes al botón **ToolBar**?

Un control ToolBar utiliza un control ImageList como fuente de imágenes. Con la propiedad ImageList del control ToolBar especificamos el control ImageList de qué obtendrá imágenes el control Toolbar.

9. Hacer clic en **Aceptar**.

✍ **Añadir imágenes a un control ToolBar utilizando un control ImageList**

1. En la ventana Propiedades de **ToolBar1**, desplazarse hacia abajo hasta visualizar la propiedad **ImageList** y, a continuación, hacer clic en **ImageList**.
2. Abrir la lista para la propiedad **ImageList** y, a continuación, hacer clic en **ImageList1**.
3. En la bandeja de componentes en la parte inferior de la vista de Diseño, hacer clic en **ImageList1**.
4. En la ventana Propiedades de **ImageList1**, hacer clic en **Images**.
5. En la columna de valores de la propiedad **Images**, hacer clic en el botón de **puntos suspensivos (...)**.
6. En el cuadro de diálogo del Editor de la colección **Image**, hacer clic en **Agregar**.
7. En la lista **Look in**, abrir pract02\Starter\bin, hacer clic en **CUT.BMP** y, a continuación, hacer clic en **Open**.
8. Repetir los pasos 6 y 7 para añadir las imágenes Copy.bmp, Paste.bmp, Delete.bmp, New.bmp y Open.bmp a **ImageList1** y, a continuación, hacer clic en **Aceptar**.
9. Abrir el Editor de colecciones ToolBarButton y, a continuación, hacer clic en **ToolBarButton1**.
10. En la ventana Propiedades de ToolBarButton1, hacer clic en **ImageIndex**, y, en la lista de valores de la propiedad **ImageIndex**, hacer clic en **image index 0** (el icono con tijeras).
11. Utilizar la tabla de propiedades de los botones 2-4 y 6-7 de ToolBar para asignar un valor a la propiedad **ImageIndex**. Cuando acabemos de añadir imágenes a **ToolBar1**, deberíamos tener el siguiente aspecto:



12. Hacer clic en **Aceptar** y guardar los cambios de nuestra aplicación.
13. Ejecutar la aplicación. ¿Ocurre algo cuando hacemos clic en un botón de la barra de herramientas? ¿Qué evento debe controlarse para responder a los clics de los botones de la barra de herramientas?

El evento ButtonClick de ToolBar se utiliza para controlar los clics de los botones de la barra de herramientas.

ToolBarButtonClickEventArgs se utiliza para determinar qué botón se ha pulsado.

14. Cerrar la aplicación.

✎ **Asignar valores a las propiedades Tag y ToolTipText de los botones de ToolBar**

15. Abrir el Editor de la colección **ToolBarButton** y, a continuación, hacer clic en **ToolBarButton1**.

16. En la tabla de propiedades de **ToolBarButton1**, hacer doble clic en **Tag**, escribir **Cortar** y presione ENTER.

17. Utilizar la tabla de propiedades de **ToolBarButton2**, **ToolBarButton3**, **ToolBarButton4**, **ToolBarButton6** y **ToolBarButton7** para asignar los valores de la propiedad **Tag** a los botones en el orden siguiente: **Copiar**, **Pegar**, **Eliminar**, **Nuevo** y **Abrir**.

18. En la tabla de propiedades de **ToolBarButton1**, hacer doble clic en **ToolTipText**, escribir **Cortar elemento** y presionar ENTER.

19. Opcional: utilizar la tabla de propiedades de **ToolBarButton2**, **ToolBarButton3**, **ToolBarButton4**, **ToolBarButton6** y **ToolBarButton7** para asignar los valores de la propiedad **ToolTipText** a los botones en el orden siguiente: **Copiar elemento**, **Pegar elemento**, **Eliminar un elemento existente**, **Crear un nuevo elemento** y **Abrir un elemento existente**.

20. Hacer clic en **Aceptar** y guardar los cambios de la aplicación.

✎ **Crear un controlador de eventos para el evento ButtonClick**

1. Abrir **Form1.vb** en el Editor de código.

2. En la lista **Nombre de clase**, hacer clic en **ToolBar1**.

3. En la lista **Nombre de método**, hacer clic en **ButtonClick**.

4. Añadir las siguientes instrucciones de código a la subrutina **ToolBar1_ButtonClick**:

```
Select Case e.Button.Tag
    Case "Cut"
        Me.cutRadioButton.Checked = True
        panelText = Me.cutRadioButton.Text & _
            " Radio Button is checked"

End Select
Me.StatusBar1.Panels(0).Text = panelText
MessageBox.Show("The " & e.Button.Tag & _
    " button is index number " & _
    ToolBar1.Buttons.IndexOf(e.Button))
```

5. En el menú **Ver**, seleccionar **Mostrar tareas** y, a continuación, hacer clic en **Comentario**.

6. En la lista de tareas, hacer doble clic en **TODO: paste inside Select Case**.

7. Utilizar la operación de cortar y pegar para trasladar el código comentado a la estructura **Select Case** que ha creado, y a continuación quite los comentarios de las instrucciones del código.

8. ¿Cuales son algunas de las razones para utilizar la propiedad **Tag** para determinar qué botón de la barra de herramientas **ToolBar** se pulsa?

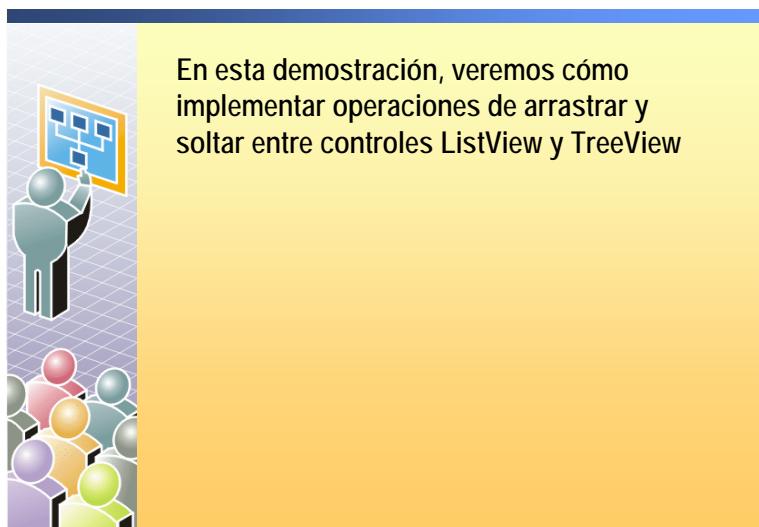
No es necesario actualizar el código cuando cambia el orden de los botones y no es necesario considerar los separadores de los botones.

9. Ejecutar la aplicación. Probar las sugerencias **ToolTips** para los botones **ToolBar** posicionando el cursor del ratón sobre un botón. Comprobar el controlador de eventos **ButtonClick** haciendo clic en los botones de **ToolBar**.

10. Si hay tiempo, examinar el código utilizado para construir el control **StatusBar**.

11. Guardar el proyecto y cerrar Visual Studio.

Demostración: implementar operaciones de arrastrar y soltar entre controles



Introducción

Otro modo de mejorar la utilidad de los controles añadidos a una aplicación es habilitar las operaciones de arrastrar y soltar. El concepto de la funcionalidad de arrastrar y soltar resulta familiar para los usuarios, y en algunos casos, incluso se espera. La forma en que implementamos la funcionalidad de arrastrar y soltar dependerá de los controles que se utilizan. La siguiente demostración muestra el código que debe añadirse a una aplicación para utilizar la capacidad de arrastrar y soltar para mover datos entre dos controles **TextBox** y entre dos controles **TreeView**.

Instrucciones

🔗 Abrir el archivo de solución SimpleDragDrop

- Abrir la solución SimpleDragDrop.sln en Visual Studio .NET desde la carpeta demo01\SimpleDragDrop que se puede encontrar dentro del fichero demos07.zip.

🔗 Ejecutar la aplicación y mostrar una operación de arrastrar y soltar entre los controles TextBox

1. Ejecutar la aplicación.
2. Posicionar el ratón sobre el control **TextBox** del lado izquierdo del formulario.
3. Presionando el botón izquierdo del ratón, arrastrar el contenido del control **TextBox** izquierdo y soltarlo sobre el **TextBox** de la parte superior derecha.
4. Observar que el cursor del ratón cambia e indica si el control está configurado para aceptar datos soltados. Este control no los aceptará.
5. Con el ratón sobre el **TextBox** de la parte superior derecha, soltar el botón izquierdo.
6. Presionando el botón izquierdo del ratón, arrastrar el contenido del control **TextBox** izquierdo a una posición sobre el control **TextBox** de la parte inferior derecha.

7. Observar que el cursor del ratón cambia para indicar que este control aceptará datos soltados.
8. Con el ratón sobre el control **TextBox** de la parte inferior derecha, soltar el botón izquierdo.
9. Cerrar la aplicación.

🔍 Revisar el código utilizado para soportar las operaciones de arrastrar y soltar entre dos controles **TextBox**

1. Abrir Form1.vb en la vista Diseño.
2. Hacer clic en el control **TextBox** de la parte superior derecha. Observar que la propiedad **AllowDrop** está configurada a *False*.
3. Hacer clic en el control **TextBox** de la parte inferior derecha. Observar que la propiedad **AllowDrop** está establecida en *True*.
4. Abrir Form1.vb en el Editor de código.
5. Examinar el controlador `TextBox1_MouseDown` que se utiliza para iniciar esta operación de arrastrar y soltar.

Observar que el método **DoDragDrop** se utiliza para especificar los datos que se utilizarán en la operación de arrastrar y soltar. El método **DoDragDrop** también se utiliza para especificar el tipo, o tipos, de operaciones permitidas. La información asociada a esta operación se almacena en la variable de argumentos del evento.

6. Examinar el controlador `TextBox2_DragEnter`.

Observar que se comprueba el tipo de datos de los datos que se arrastran para garantizar que los datos pueden utilizarse según lo previsto. El parámetro *DragEventArgs* se utiliza para acceder a los datos y, si los datos son de un tipo inadecuado, la propiedad **DragEventArgs.Effect** se establece como **DragDropEffects.None**.

7. Examinar el controlador `TextBox2_DragDrop`.

Observar que los datos que contiene el método **GetDatun** de la propiedad **DragEventArgs.Data** se utilizan para extraer los datos que se insertarán en el control.

🔍 Ejecutar la aplicación y mostrar una operación de arrastrar y soltar entre controles **TreeView**

1. Ejecutar la aplicación.
2. Posicionar el ratón sobre el nodo *Mustard* que se muestra en el control **TreeView** izquierdo.
3. Presionando el botón izquierdo del ratón, arrastrar el nodo *Mustard* a una posición sobre el nodo *Garnishes* del control **TreeView** derecho.
4. Con el ratón sobre el nodo *Garnishes*, soltar el botón izquierdo.
5. Cerrar la aplicación.

✍ **Examinar el código que se utiliza para soportar operaciones de arrastrar y soltar entre dos controles `TreeView`**

1. Abrir `Form1.vb` en el Editor de código.
2. Examinar el controlador **`TreeView_ItemDrag`** que se utiliza para iniciar esta operación de arrastrar y soltar.

Observar que el control **`TreeView`** incluye un evento especial diseñado para iniciar una operación de arrastrar y soltar. El método **`DoDragDrop`** se utiliza de nuevo para especificar los datos (en este caso, el elemento seleccionado actualmente) que se utilizarán en la operación de arrastrar y soltar si el tipo de operación está permitido.

3. Examinar el controlador **`TreeView_DragEnter`**.

Observar que el controlador de eventos **`DragEnter`** se utiliza del mismo modo en que se ha utilizado en la operación de arrastrar y soltar entre controles **`TextBox`**.

4. Examinar el controlador **`TreeView_DragDrop`**.

Observar cómo están controlados **`TreeNode`**s en este procedimiento.

5. Cerrar Visual Studio .NET.

Lección: uso de los cuadros de diálogo en una aplicación Windows Forms

- Seleccionar cuadros de diálogo en Visual Studio .NET
- Cómo mostrar cuadros de diálogo en una aplicación
- La propiedad DialogResult
- Cómo utilizar la entrada en los cuadros de diálogo
- Demostración: uso del control OpenFileDialog

Introducción

Los cuadros de diálogo se utilizan para interactuar con el usuario y recuperar datos introducidos por el mismo. Visual Studio .NET proporciona algunos cuadros de diálogo preconfigurados que pueden utilizarse en las aplicaciones Windows Forms para interactuar con los usuarios. Esta lección presenta los diálogos estándares que proporciona Visual Studio .NET y explica cómo utilizar estos cuadros de diálogo para recuperar los datos introducidos por el usuario.

Objetivos de la lección

En esta lección, estudiaremos cómo:

- Seleccionar cuadros de diálogo Visual Studio .NET apropiados para una aplicación Windows Forms.
- Utilizar los cuadros de diálogo disponibles en Visual Studio .NET en una aplicación Windows Forms.
- Recuperar la entrada del usuario utilizando la propiedad **DialogResult**.

Seleccionar cuadros de diálogo en Visual Studio .NET

OpenFileDialog	Permite a los usuarios abrir archivos mediante un cuadro de diálogo preconfigurado
SaveFileDialog	Selecciona los archivos a guardar y la ubicación donde deben guardarse
ColorDialog	Permite a los usuarios seleccionar un color de la paleta y agregar colores a ésta
FontDialog	Expone las fuentes actualmente instaladas en el sistema
PrintDialog	Selecciona una impresora y determina otras configuraciones relacionadas con la impresión
PageSetupDialog	Configura los detalles de la página para su impresión
PrintPreviewDialog	Muestra el aspecto que tendrá un documento cuando se imprima

Introducción Visual Studio .NET incluye un conjunto de cuadros de diálogo preconfigurados que podemos adaptar para nuestras propias aplicaciones. Dependiendo de los requerimientos de la aplicación, podemos seleccionar el cuadro de diálogo apropiado de uno de los cuadros de diálogo preconfigurados.

OpenFileDialog Podemos utilizar el control **OpenFileDialog** en una aplicación Windows Forms como una solución sencilla para seleccionar archivos en lugar de configurar nuestro propio cuadro de diálogo. Cuando utilizamos el control **OpenFileDialog**, deberemos escribir nuestra propia lógica para abrir archivos. **OpenFileDialog** es el mismo cuadro de diálogo **Archivo Abrir** utilizado en el sistema operativo Microsoft Windows®. Cuando se añade a un formulario, el control **OpenFileDialog** aparece en la bandeja de la parte inferior del Diseñador de Windows Forms. Hereda de la clase **CommonDialog**.

SaveFileDialog El control **SaveFileDialog** permite a los usuarios guardar archivos en una aplicación. Al igual que el resto de cuadros de diálogo, cuando utilizamos el control **SaveFileDialog**, deberemos escribir nuestra propia lógica para guardar archivos. **SaveFileDialog** es el mismo que el cuadro de diálogo **Guardar archivo** estándar utilizado por Windows. Hereda de la clase **CommonDialog**.

ColorDialog El control **ColorDialog** de Windows Forms es un cuadro de diálogo preconfigurado que permite al usuario seleccionar un color de una paleta y añadir colores personalizados a dicha paleta. Es el mismo cuadro de diálogo que podemos ver en otras aplicaciones Windows para seleccionar colores.

El color seleccionado en el cuadro de diálogo se devuelve en la propiedad **Color**. Si la propiedad **AllowFullOpen** está establecida en **False**, el botón **Define Custom Colors** está deshabilitado y el usuario está limitado a los colores predefinidos en la paleta. Si la propiedad **SolidColorOnly** está establecida en **True**, el usuario no puede seleccionar colores neutros.

FontDialog	El control FontDialog de Windows Forms es el cuadro de diálogo Font de Windows estándar utilizado para exponer las fuentes actualmente instaladas en el sistema. De forma predeterminada, el cuadro de diálogo incluye opciones para Fuente, Estilo de la fuente y Tamaño. También incluye casillas de verificación para efectos como Tachado y Subrayado, y una lista despegable para <i>Script</i> .
PrintDialog	El control PrintDialog se utiliza para seleccionar una impresora, determinar las páginas a imprimir y demás configuraciones relacionadas con la impresión de aplicaciones Windows. Podemos proporcionar a los usuarios opciones como imprimir todo, imprimir un intervalo específico de páginas o imprimir una selección.
PageSetupDialog	<p>El control PageSetupDialog se utiliza para configurar la página de impresión en las aplicaciones Windows. Podemos permitir a los usuarios que establezcan los ajustes de bordes y márgenes, encabezados y pies, y la orientación vertical u horizontal.</p> <p>El control PageSetupDialog permite a los usuarios establecer propiedades relacionadas con una sola página (clase PrintDocument) o cualquier documento (clase PageSettings). Además, el control PageSetupDialog puede utilizarse para determinar configuraciones específicas de la impresora, que se almacenan en la clase PrinterSettings.</p>
PrintPreviewDialog	El control PrintPreviewDialog se utiliza para mostrar qué aspecto tendrá un documento cuando se imprima. El control contiene botones para imprimir, hacer zoom, mostrar una o múltiples páginas y cerrar el cuadro de diálogo.

Cómo mostrar cuadros de diálogo en una aplicación

■ Mostrar un cuadro de diálogo Visual Studio .NET preconfigurado

```
Private Sub Button1_Click(ByVal sender as
    System.Object,ByVal e as System.EventArgs)
    OpenFileDialog1.ShowDialog()
End Sub
```

■ Mostrar un cuadro de diálogo de mensaje

```
Private Sub PerformSearch()
    MessageBox.Show("The search is now complete", _
        "My Application", MessageBoxButtons.OKCancel, _
        MessageBoxIcon.Asterisk)
End Sub
```

Introducción

Mostramos cuadros de diálogo en una aplicación del mismo modo que mostramos un formulario. Para mostrar cualquiera de los cuadros de diálogo de Visual Studio .NET en una aplicación, debemos escribir el código para cargarlo y mostrarlo, del mismo modo que mostraríamos un segundo formulario en el formulario principal.

Procedimiento: mostrar un cuadro de diálogo preconfigurado

Para mostrar un cuadro de diálogo preconfigurado:

1. En el Editor de código, ir al controlador de eventos con el que desea abrir el cuadro de diálogo.

El primer paso es localizar el controlador de eventos que se utilizará para mostrar el cuadro de diálogo. En una aplicación, un cuadro de diálogo normalmente se abre en respuesta al clic de un botón o a un comando de menú, pero se puede utilizar cualquier evento.

2. Añadir el código para mostrar el cuadro de diálogo.

Utilizar el método **ShowDialog** para mostrar un cuadro de diálogo en las aplicaciones Windows Forms.

```
Private Sub Button1_Click(ByVal sender as
    System.Object,ByVal e as System.EventArgs)
    OpenFileDialog1.ShowDialog()
End Sub
```

Procedimiento: mostrar un cuadro de mensaje

Podemos mostrar un cuadro de mensaje utilizando el método **Show** de la clase **MessageBox**. El método **Show** requiere que suministremos el texto del mensaje y, opcionalmente, podemos especificar el título del cuadro de diálogo, los botones, el icono, el botón predeterminado y opciones relacionadas con el modo en que se mostrarán el cuadro de mensaje y el texto que contenga.

```
Private Sub PerformSearch()  
    MessageBox.Show("The search is now complete", _ "My  
Application", MessageBoxButtons.OKCancel, _  
    MessageBoxIcon.Asterisk)  
End Sub
```

Propiedad DialogResult

Propiedad DialogResult

Utilizar el valor devuelto por esta propiedad para determinar qué acción ha realizado el usuario

Ejemplo

El valor **DialogResult.Cancel** indica que el usuario ha hecho clic en el botón **Cancelar**

La propiedad **DialogResult** puede establecerse en tiempo de diseño o en tiempo de ejecución

Introducción	<p>Cuando mostramos un cuadro de diálogo en una aplicación, es muy importante saber la entrada resultante. Por ejemplo, si mostramos un cuadro de diálogo que indica al usuario que acepte o cancele una acción, debemos saber si hace clic en el botón Aceptar o en el botón Cancelar.</p>
La propiedad DialogResult	<p>La entrada del usuario en un cuadro de diálogo la procesa el formulario que muestra el cuadro de diálogo. Podemos utilizar la propiedad DialogResult del formulario para determinar el resultado de la entrada del usuario. En función del valor devuelto por la propiedad DialogResult, podemos decidir desechar o utilizar la información devuelta por el cuadro de diálogo.</p>
Ejemplo	<p>Si un usuario hace clic en el botón Cancelar de un cuadro de diálogo, el valor de la propiedad DialogResult se establece en DialogResult.Cancel. El formulario primario recupera este valor y deshecha la información del cuadro de diálogo.</p>
Establecer la propiedad DialogResult en tiempo de diseño o en tiempo de ejecución	<p>Podemos establecer la propiedad DialogResult en tiempo de diseño o en tiempo de ejecución. En tiempo de diseño, podemos establecer la propiedad DialogResult para todos los controles de botón del cuadro de diálogo. Establecer la propiedad DialogResult en tiempo de ejecución permite controlar dinámicamente las respuestas de los usuarios.</p>

Cómo utilizar la entrada de los cuadros de diálogo

Recuperar y utilizar resultados de cuadros de diálogo

- 1** En la ventana de código, ir al controlador de eventos o el método para el que desea establecer la propiedad **DialogResult**
- 2** Añadir código para recuperar el valor **DialogResult**

```
Dim userResponse As DialogResult =
    OpenFileDialog1.ShowDialog()
If userResponse = DialogResult.OK Then
    filePath = OpenFileDialog1.FileName.ToString
    MessageBox.Show("You successfully opened: '" &
        filePath & "'", "Success", MessageBoxButtons.OK,
        MessageBoxIcon.Information,
        MessageBoxDefaultButton.Button1)
```

Introducción

Después de que se cierra el cuadro de diálogo, el formulario responsable de mostrar el cuadro puede hacer referencia a los valores con la propiedad **DialogResult**.

Procedimiento

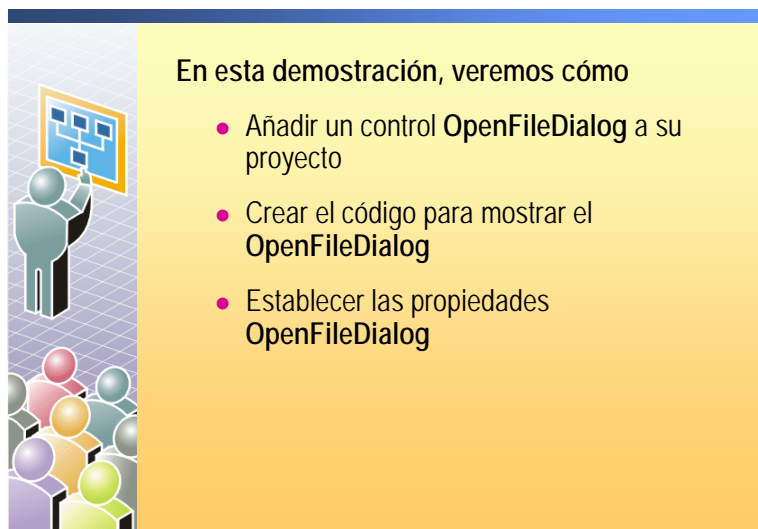
Para recuperar y utilizar la entrada de un cuadro de diálogo:

1. En el Editor de código, ir al controlador de eventos o al método para el que deseamos establecer la propiedad **DialogResult**.
2. Añadir código para recuperar el valor **DialogResult**.

El siguiente código muestra cómo se recupera la entrada del usuario de un cuadro de diálogo **Abrir Archivo**.

```
Dim userResponse As DialogResult =
    OpenFileDialog1.ShowDialog()
If userResponse = DialogResult.OK Then
    filePath = OpenFileDialog1.FileName.ToString
    MessageBox.Show("You successfully opened: '" _
        & filePath & "'", "Success", _
        MessageBoxButtons.OK, _
        MessageBoxIcon.Information, _
        MessageBoxDefaultButton.Button1)
Else
    MessageBox.Show("You canceled the open _
        file operation.", "Warning", _
        MessageBoxButtons.OK, _
        MessageBoxIcon.Warning, _
        MessageBoxDefaultButton.Button1, _
        MessageBoxOptions.RightAlign)
End If
```

Demostración: uso del control OpenFileDialog



Introducción

En esta demostración, veremos cómo utilizar el control **OpenFileDialog** en una aplicación Windows Forms.

Instrucciones

🔍 Abrir el archivo OpenFileDialog.sln

- Abrir la solución OpenFileDialog.sln en Visual Studio .NET desde la carpeta demo02\Starter que se puede encontrar dentro del fichero demos07.zip. La solución del ejercicio se encuentra en la carpeta demo02\Solution dentro del mismo fichero comprimido.

🔍 Añadir un control OpenFileDialog al proyecto

1. Abrir Form1.vb en la vista Diseño.
2. En el cuadro de herramientas, hacer doble clic en **OpenFileDialog**.

🔍 Crear el código para mostrar el OpenFileDialog

1. Abrir Form1.vb en el Editor de código.
2. En el menú **Ver**, seleccionar **Mostrar tareas** y, a continuación, hacer clic en **Comentario**.
3. En la Lista de tareas, hacer doble clic en **TODO: show the OpenFileDialog y check DialogResult**.
4. Añadir la siguiente instrucción de código debajo de la línea TODO:


```
OpenFileDialog1.ShowDialog
```
5. Ejecutar la aplicación y, a continuación, hacer clic en **Use the OpenFileDialog Control**.
6. Hacer clic en **Cancelar** y cerrar la aplicación.
7. ¿Qué podemos hacer para determinar si el cuadro de diálogo se ha cerrado haciendo clic en **Abrir** o en **Cancelar**?

Podemos utilizar la propiedad DialogResult del formulario primario.

--

8. Sustituir la instrucción del código 'OpenFileDialog1.ShowDialog' por las siguientes sentencias de código:

```
If OpenFileDialog1.ShowDialog = DialogResult.OK Then
    filePath = OpenFileDialog1.FileName.ToString
    MessageBox.Show("You successfully opened: '" & _
        filePath & "'", _
        "Success", _
        MessageBoxButtons.OK, _
        MessageBoxIcon.Information, _
        MessageBoxDefaultButton.Button1)

Else
    MessageBox.Show("You canceled the operation.", _
        "Warning", _
        MessageBoxButtons.OK, _
        MessageBoxIcon.Warning, _
        MessageBoxDefaultButton.Button1, _
        MessageBoxOptions.RightAlign)

End If
```

9. Ejecutar la aplicación y, a continuación, hacer clic en **Use the OpenFileDialog Control**.
10. Hacer clic en **Cancelar** y cerrar la aplicación.

✍ Establecer las propiedades OpenFileDialog

1. En la Lista de tareas, hacer doble clic en **TODO: set the initial directory and filter properties**.
2. Añadir la siguiente instrucción de código debajo de la línea TODO:

```
OpenFileDialog1.InitialDirectory = Application.StartupPath
OpenFileDialog1.Filter = "Text Files (*.txt)|*.txt"
```

3. Ejecutar la aplicación y, a continuación, hacer clic en **Use the OpenFileDialog Control**.
4. Utilizar **Abrir** y **Cancelar** para cerrar el cuadro de diálogo, y cerrar la aplicación.
5. ¿Qué ocurre cuando seleccionamos un archivo y hacemos clic en **Abrir**?
Las propiedades de OpenFileDialog se actualizan para reflejar el archivo que ha seleccionado. Podemos utilizar esta información para realizar acciones en el archivo seleccionado, como mostrar su contenido utilizando un PrintPreviewDialog.

6. Si disponemos de tiempo, experimentar las propiedades **OpenFileDialog** adicionales, como **Multiselect** y **CheckPathExists**, y modificar el aspecto de los cuadros de diálogo de mensajes.
7. Guardar el proyecto y cerrar Visual Studio .NET.

Lección: agregar controles en tiempo de ejecución

- Colección de controles
- Cómo agregar controles en tiempo de ejecución
- Práctica: agregar y eliminar controles en tiempo de ejecución

Introducción

Visual Studio .NET ofrece la flexibilidad de agregar controles en tiempo de ejecución. Esta lección presenta la colección de controles y cómo podemos añadir controles en tiempo de ejecución.

Objetivos de la lección

En esta lección, estudiaremos cómo:

- Utilizar la propiedad **Controls** para acceder al objeto **ControlCollection** de un contenedor.
- Añadir y eliminar controles de un contenedor en tiempo de ejecución.

Colección de controles

■ Colección de controles

- Representa un colección de objetos de control
- Utilizar los métodos **Add**, **Remove** y **RemoveAt** para agregar y eliminar controles de la colección

```
Form1.Controls.Add(textbox1)
```

```
Form1.Controls.Remove(textbox1)
```

- Utilizar el método **Contains** para determinar si un control forma parte o no de una colección

```
Form1.Controls.Contains(textbox1)
```

Introducción

Visual Studio .NET ofrece un objeto de colección que contiene todos los controles de un formulario o de un control contenedor. Este objeto se denomina **ControlCollection** y se accede a él utilizando la propiedad **Controls** del formulario o control. La colección de controles representa una colección de objetos *Control*. Podemos añadir y eliminar controles de un contenedor en tiempo de ejecución utilizando la propiedad **Controls**. La propiedad **ControlCollection Count** devuelve el número de controles del contenedor y puede utilizarse para recorrer cíclicamente la colección de controles.

Métodos ControlCollection

La siguiente tabla ofrece una lista de algunos de los métodos de **ControlCollection**:

Método	Descripción
Add	Agrega el control especificado a la colección de controles.
AddRange	Agrega una matriz de objetos de control a la colección.
Clear	Elimina todos los controles de la colección.
Contains	Determina si el control especificado es miembro de la colección.
Remove	Elimina el control especificado de la colección de controles.
RemoveAt	Elimina un control de la colección de controles en la ubicación indexada especificada.
ToString (heredado desde Objeto)	Devuelve una cadena (<i>string</i>) que representa el Objeto actual.
IndexOf	Recupera el índice del control especificado en la colección de controles.
GetEnumerator	Devuelve un enumerador que puede utilizarse para iterar por la colección de controles.

Cómo agregar controles en tiempo de ejecución

Para agregar controles en tiempo de ejecución

- 1 Crear el control que se agregará al contenedor

```
Dim signatureCheckBox As New CheckBox()
' set properties
signatureCheckBox.Left = 24
signatureCheckBox.Top = 80
signatureCheckBox.Text = "Signature required"
```

- 2 Añadir el control al contenedor utilizando el método **Add** de la propiedad **Controls**

```
' add the new control to the collection
GroupBox1.Controls.Add(signatureCheckBox)
```

Introducción

Podemos añadir y eliminar controles en tiempo de ejecución utilizando la propiedad **Controls**. La capacidad de añadir controles en tiempo de ejecución permite personalizar la aplicación en función de la entrada del usuario. Por ejemplo, podemos hacer que la aplicación muestre elementos de menú adicionales dependiendo de la entrada del usuario.

Procedimiento

Para añadir controles en tiempo de ejecución:

1. Crear un control que se añadirá al formulario en tiempo de ejecución.

El siguiente código muestra un ejemplo de cómo crear un control **CheckBox**.

```
Dim signatureCheckBox As New CheckBox()
```

2. Añadir el control al formulario o al control contenedor utilizando el método **Add** de la propiedad **Controls**.


El siguiente código añade el control de botón en una ubicación específica del formulario.

```
signatureCheckBox.Name = "myCheckBox"
signatureCheckBox.Text = "Signature required"
signatureCheckBox.Width = 224
signatureCheckBox.Left = 24
signatureCheckBox.Top = 80

GroupBox1.Controls.Add(signatureCheckBox)
```

Nota Cuando se añaden varios controles a un control primario, es recomendable invocar el método **SuspendLayout** antes de inicializar los controles que van a añadirse. Después de añadirlos al control primario, invocar al método **ResumeLayout**. De esta forma, se incrementará el rendimiento de las aplicaciones con muchos controles.

Práctica: agregar y eliminar controles en tiempo de ejecución



En esta práctica,

- Eliminaremos los controles no deseados
- Agregaremos un nuevo control
- Especificaremos propiedades del nuevo control

Empezar examinando los objetivos de esta actividad práctica

8 min

Introducción

En esta práctica, agregaremos y eliminaremos controles en tiempo de ejecución.

Instrucciones

🔍 Abrir el proyecto de la práctica

1. Utilizando Windows Explorer, ir a pract03\Starter. Esta carpeta se puede encontrar dentro del fichero practs07.zip. La solución del ejercicio práctico se puede encontrar en la carpeta pract03\Solution dentro del mismo fichero comprimido.
2. Hacer doble clic en el archivo de solución **AddingAndRemovingControls.sln** para abrir el proyecto.

🔍 Eliminar controles no deseados

1. Abrir Form1.vb en el Editor de código.
2. En el menú **Ver**, seleccionar **Mostrar tareas** y hacer clic en **Comentario**.
3. En la Lista de tareas, hacer doble clic en **TODO: remove controls from GroupBox1**.
4. Añadir las siguientes instrucciones de código debajo de la línea **TODO**:

```
1cv = GroupBox1.Controls.Count
Do While 1cv > 2
    GroupBox1.Controls.Remove(GroupBox1.Controls(1cv - 1))
    1cv -= 1
```

Loop

5. Ejecutar la aplicación, hacer clic en **Use International Policies** y, a continuación, hacer clic en **Use Domestic Policies**.
6. Cerrar la aplicación.

✍ Añadir un nuevo control

1. En la Lista de tareas, hacer doble clic en **TODO: create an instance of a CheckBox**.

2. Añadir la siguiente instrucción de código debajo de la línea TODO:

```
Dim decimalCheckBox As New CheckBox()
```

3. En la Lista de tareas, hacer doble clic en **TODO: add a control to GroupBox1**.

4. Añadir la siguiente instrucción de código debajo de la línea TODO:

```
GroupBox1.Controls.Add(decimalCheckBox)
```

5. Ejecutar la aplicación, hacer clic en **Use International Policies** y, a continuación, hacer clic en **Use Domestic Policies**.

6. ¿Por qué aparece **decimalCheckBox** en la esquina superior izquierda de **GroupBox1**?

Los valores predeterminados de la propiedad de la ubicación de un control son cero para ambas coordenadas, X e Y.

-
7. Cerrar la aplicación.

✍ Especificar las propiedades del nuevo control

1. En la Lista de tareas, hacer doble clic en **TODO: specify the control properties**.

2. Añadir la siguiente instrucción de código debajo de la línea TODO:

```
decimalCheckBox.Left = 24  
decimalCheckBox.Top = _  
    GroupBox1.Controls(GroupBox1.Controls.Count - 1).Top _  
    + 32  
decimalCheckBox.Text = "Use comma separated decimals"  
decimalCheckBox.Name = "commaSeparatedDecimalsCheckBox"
```

3. Ejecutar la aplicación, hacer clic en **Use International Policies** y, a continuación, hacer clic en **Use Domestic Policies**.

4. Cerrar la aplicación.

5. En la Lista de tareas, hacer doble clic en **TODO: specify the control properties**.

6. Añadir la siguiente instrucción de código debajo de la línea TODO:

```
decimalCheckBox.Width = 224
```

7. Ejecutar la aplicación, hacer clic en **Use International Policies** y, a continuación, hacer clic en **Use Domestic Policies**.

8. Cerrar la aplicación.

9. Si disponemos de tiempo, abrir Form1.vb en la vista Diseño y seguir estos pasos:
 - a. Arrastrar **Use local dataset when available CheckBox** desde **GroupBox1** a **GroupBox2** y, a continuación, arrastrarlo de nuevo a su posición original en **GroupBox1**.
Éste es ahora el último control añadido a **GroupBox1**.
 - b. Ejecutar la aplicación y probar su código haciendo clic en los botones de opción.
 - c. Eliminar **Print report automatically** de **GroupBox1** y sustituirlo.
Print report automatically es ahora el último control en la **ControlCollection** de **GroupBox1**.
 - d. Probar la aplicación de nuevo para comprobar si nuestro código añade controles a **GroupBox1** en las ubicaciones deseadas.
10. Guardar el proyecto y cerrar Visual Studio .NET.

Lección: crear menús

- Cómo agregar un menú contextual a un formulario
- Cómo agregar elementos de menú en tiempo de ejecución
- Cómo crear controladores de menú para elementos de menú
- Cómo utilizar propiedades de menú
- Práctica: actualizar menús en tiempo de ejecución

Introducción

Los menús y los menús contextuales son formas de exponer funcionalidad a nuestros usuarios o alterarles de información importante de nuestra aplicación. Los menús contienen comandos, agrupados por un tema común. Los menús contextuales emergen en respuesta a un clic del botón secundario del ratón y contienen comandos utilizados habitualmente para un área concreta de una aplicación.

Nota Podemos tener múltiples menús principales para una aplicación. Si nuestra aplicación tiene un gran tamaño, podemos utilizar diferentes menús para diferentes partes de una aplicación.

Objetivos de la lección

En esta lección, estudiaremos cómo:

- Crear menús contextuales.
- Añadir elementos de menú en tiempo de ejecución.
- Crear accesos directos a menús.
- Habilitar la propiedad *checked* para elementos de menú.

Cómo agregar un menú contextual a un formulario

Para añadir controles en tiempo de ejecución

1 En la Caja de herramientas, hacer doble clic en el control **ContextMenu**

2 Asociar el menú contextual a un formulario o a un control estableciendo la propiedad **ContextMenu** de ese objeto

Para agregar un menú contextual programáticamente

```
Public Sub AddContextMenu()
    Dim ctxmenu as New ContextMenu()
    Me.ContextMenu() = ctxmenu
    ...
End Sub
```

Introducción

El control **ContextMenu** de Windows Forms se utiliza para proporcionar a los usuarios un menú fácilmente accesible de los comandos utilizados frecuentemente que están asociados al objeto seleccionado. A menudo, los elementos de un menú contextual son un subconjunto de los elementos de menús principales que aparecen en otra parte de la aplicación. Normalmente, se accede a los menús contextuales haciendo clic con el botón derecho del ratón. En Windows Forms, están asociados a controles.

Procedimiento: agregar un menú contextual en tiempo de diseño

Para agregar un menú contextual en tiempo de diseño:

1. Añadir el menú contextual desde la barra de herramientas al formulario.
En el cuadro de herramientas, hacer doble clic en el control **ContextMenu**.
2. Asociar el menú contextual a un formulario o a un control estableciendo la propiedad **ContextMenu** de ese objeto.
Podemos asociar un menú contextual a un control estableciendo la propiedad **ContextMenu** del control al control **ContextMenu**. Podemos asociar un único menú contextual a múltiples controles, pero cada control puede tener sólo un menú contextual.

Procedimiento: agregar un menú contextual en tiempo de diseño

Para añadir un menú contextual en tiempo de diseño:

1. Crear un nuevo método que incluya el código requerido para crear un elemento de menú.

```
Public Sub AddContextMenu()
    ...
End Sub
```

2. Crear una instancia del menú contextual.

Añadir el código como se muestra en el ejemplo para crear una instancia del menú contextual.

```
Dim ctxmenu as New ContextMenu()
Me.ContextMenu() = ctxmenu
```

Cómo añadir elementos de menú en tiempo de ejecución

Cómo agregar elementos de menú en tiempo de ejecución

Agregar elementos de menú a un menú contextual en tiempo de ejecución	
1	Dentro del método, crear objetos MenuItem para añadirlos al menú contextual de la colección Object <code>Dim menuItemNew as New MenuItem()</code>
2	Dentro del método, establecer la propiedad Text para cada elemento de menú <code>MenuItemNew.Text = "New"</code>
3	Dentro del método, añadir elementos de menú a la colección MenuItems del objeto ContextMenu <code>ctxMenu.MenuItems.Add(menuItemNew)</code>

Introducción

Después de añadir un control **ContextMenu** a su formulario, podemos añadir elementos de menú. El contenido de los menús se almacena en una colección. Podemos añadir elementos de menú a un menú contextual en tiempo de ejecución agregando objetos **MenuItem** a esta colección.

Procedimiento

La principal propiedad del control **ContextMenu** es la propiedad **MenuItems**. Podemos añadir elementos de menú programáticamente creando objetos **MenuItem** y añadiéndolos a la colección **MenuItems** del menú contextual. Debido a que los elementos de un menú contextual normalmente se arrastran desde otros menús, frecuentemente añadiremos elementos a un menú contextual copiándolos. También podemos deshabilitar, ocultar o eliminar elementos de menú.

Para añadir elementos a un menú contextual en tiempo de ejecución:

1. En el método para crear un menú contextual, crear objetos **MenuItem** que se añaden a la colección de controles.

```
Dim menuItemNew as New MenuItem()
```

2. Establecer la propiedad **Text** para cada elemento de menú.

```
MenuItemNew.Text = "New"
```

3. Añadir elementos de menú a la colección **MenuItems** del objeto **ContextMenu**.

```
ctxMenu.MenuItems.Add(menuItemNew)
```

Nota También podemos añadir imágenes a elementos de menú. Para ello, crear una instancia de la clase **MenuItem**, sobrecargar el método **OnPaint()**, y arrastrar la imagen junto al texto del elemento de menú.

Cómo crear controladores de eventos para elementos de menú

Para agregar funcionalidades a los elementos de menú

- 1 Crear un controlador de eventos para el evento `MenuItem.Click`

```
Private Sub MenuItemNew_Click (ByVal sender as
System.Object,ByVal e as System.EventArgs)
End Sub
```
- 2 Escribir el código para controlar el evento

```
Private Sub MenuItemNew_Click (ByVal sender as
System.Object,ByVal e as System.EventArgs)
    MessageBox.Show("You clicked the New
Option")
End Sub
```

Introducción Una vez establecida la estructura de un menú, se proporciona funcionalidad. Para ello, crearemos un controlador de eventos para el evento **MenuItem.Click** y escribiremos código para controlar el evento.

Procedimiento Para añadir funcionalidades a elementos de menú:

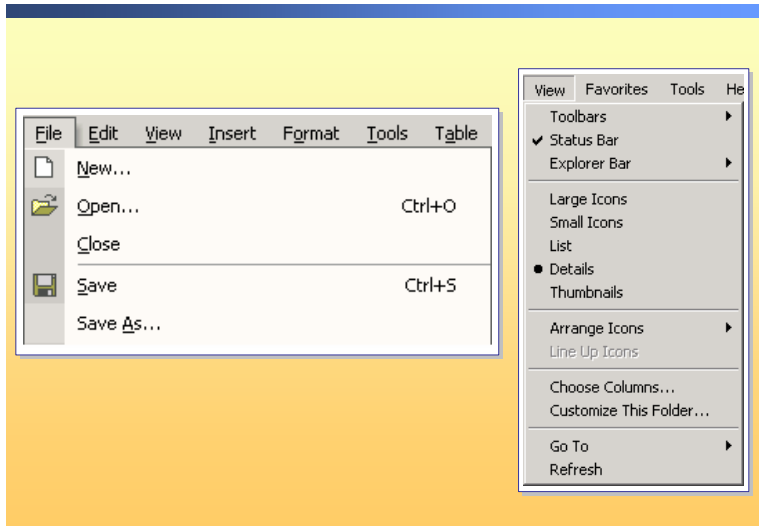
1. Crear un controlador de eventos para el evento **MenuItem.Click**.

```
Private Sub MenuItemNew_Click (ByVal sender As
System.Object,ByVal e as System.EventArgs)
End Sub
```

2. Escribir el código para controlar el evento **MenuItem.Click**.

```
Private Sub MenuItemNew_Click (ByVal sender As
System.Object,ByVal e as System.EventArgs)
    MessageBox.Show("You clicked the New Option")
End Sub
```

Cómo utilizar las propiedades del menú



Introducción

Los elementos de menú incluyen varias propiedades que permiten mejorar la experiencia del usuario. Estas propiedades incluyen propiedades para crear teclas de acceso directo y habilitar y deshabilitar elementos de menú utilizando marcas de verificación, botones de opción, etc.

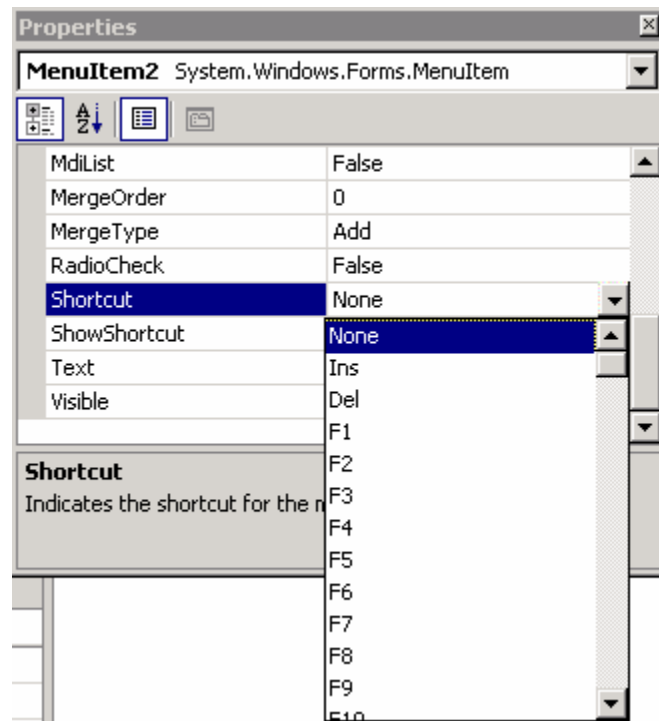
Procedimiento: crear teclas de acceso directo

Las teclas de acceso directo proporcionan a los usuarios un método para activar elementos de menú utilizados frecuentemente en el sistema de menús y proporcionar acceso a la aplicación desde el teclado.

Por ejemplo, en Microsoft Word, podemos abrir un archivo utilizando el acceso directo Ctrl+A o guardar un archivos presionando Ctrl+G.

Para crear un acceso directo a elementos de menú:

1. Seleccionar el elemento de menú para el que necesitamos crear un acceso directo.
2. En el cuadro de diálogo **Propiedades**, seleccionar la propiedad **Shortcut**.



3. Seleccionar el acceso directo requerido en la lista de accesos directos.
4. En el cuadro de diálogo **Propiedades**, establecer la propiedad **ShowShortcut** en **True**.

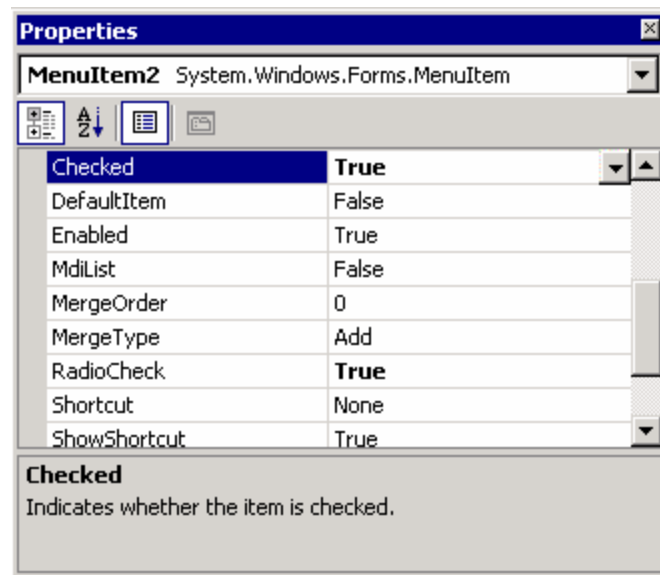
Procedimiento: mostrar
elementos de menú
seleccionados

Podemos utilizar la propiedad **Checked** y **RadioCheck** para identificar el elemento de menú seleccionado de un grupo de elementos de menú mutuamente excluyentes. También podemos colocar una marca de selección en el elemento de menú de un grupo de elementos para identificar el tamaño de la fuente que se mostrará para el texto de una aplicación.

Por ejemplo, elementos mutuamente excluyentes del menú **Ver** en Windows Explorer utilizan una selección o un botón de opción para mostrar la opción seleccionada.

Para mostrar elementos de menú seleccionados:

1. Seleccione el elemento de menú.
2. En el cuadro de diálogo **Propiedades**, establecer la propiedad **Checked** en **True**.



Para mostrar la selección de opción para los elementos de menú:

1. Seleccione el elemento de menú.
2. En el cuadro de diálogo **Propiedades**, establecer la propiedad **Checked** en **True**.
3. Establecer la propiedad **RadioCheck** en **True**.


Procedimiento: habilitar y deshabilitar elementos de menú

Es posible que deseemos deshabilitar determinados elementos de menú para usuarios dependiendo de su función, permisos o de su entrada. Podemos utilizar la propiedad **Enabled** para habilitar o deshabilitar elementos de menú. Si el valor de la propiedad **Enabled** se establece en **True**, el elemento de menú se habilita. Sin embargo, si el valor se establece en **False**, el elemento de menú se deshabilita.

Para habilitar o deshabilitar elementos de menú:

1. Seleccionar el elemento de menú.
2. Establecer la propiedad **Enabled** en **True** o **False**.


Práctica: actualizar menús en tiempo de ejecución



En esta práctica,

- Utilizaremos un segundo control **MainMenu**
- Asignaremos un **ContextMenu** a un control
- Identificaremos el control asociado a un **ContextMenu**
- Agregaremos un elemento de menú en tiempo de ejecución
- Utilizaremos la propiedad **Checked** de un elemento de menú

Empezar examinando los objetivos de esta actividad práctica



Introducción

En esta práctica, actualizaremos los menús de una aplicación en tiempo de ejecución.

Instrucciones

🔍 Abrir el proyecto de la práctica

1. Utilizando Windows Explorer, ir a pract04\Starter. Esta carpeta se puede encontrar dentro del fichero practs07.zip. La solución del ejercicio práctico se puede encontrar en la carpeta pract04\Solution dentro del mismo fichero comprimido.
2. Hacer doble clic en el archivo de solución **RuntimeMenus.sln** para abrir el proyecto.

🔍 Utilizar un segundo MainMenu

1. Ejecutar la aplicación, abrir el menú **File** y, a continuación, abrir el menú **View**.
2. En la ficha **Reports**, abrir el menú **File**, abrir el menú **View** y cerrar la aplicación.
Observar que los elementos de menú no se corresponden con los requerimientos de la ficha **Reports**.
3. En el Explorador de soluciones, hacer clic en **Form1** y, a continuación, hacer clic en **Ver código**.
4. En el menú **Ver**, seleccione **Mostrar tareas** y, a continuación, hacer clic en **Comentario**.
5. En la Lista de tareas, hacer doble clic en **TODO: assign a different MainMenu to a form**.
6. Añadir la siguiente instrucción de código debajo de la línea **TODO**:

`Me.Menu = MainMenu2`
7. Ejecutar la aplicación, abrir el menú **File** y, a continuación, abrir el menú **View**.

8. Hacer clic en la ficha **Reports**, abrir el menú **File**, y volver a abrir el menú **View**.
9. ¿Cuándo utilizaríamos más de un control **MainMenu** en una aplicación?

Puede resultar útil utilizar (mostrar) más de un objeto MainMenu cuando el contexto de nuestra aplicación cambia, o cuando nuestra aplicación tiene múltiples estados.

10. Cerrar el menú **View** y, a continuación, cerrar la aplicación.

✍️ **Asignar un ContextMenu a un control**

1. En la Lista de tareas, hacer doble clic en **TODO: create a control and assign a contextual menu to it**.
2. Añadir las siguientes instrucciones de código debajo de la línea **TODO**:

```
Dim fileLabel1 As New Label()  
fileLabel1.Dock = DockStyle.Top  
fileLabel1.Text = Application.StartupPath & "\Chai.txt"  
fileLabel1.ContextMenu = ContextMenu1  
Me.Panel1.Controls.Añadir(fileLabel1)
```

3. Ejecutar la aplicación, hacer clic con el botón derecho en el control **Label** que muestra la ruta de archivo Chai.txt y, a continuación, hacer clic en **Open**.
 4. Hacer clic en **OK**, y cerrar la aplicación.
 5. ¿Cómo podemos identificar qué control está asociado a un **ContextMenu**?
La propiedad ContextMenu.SourceControl obtiene el control mostrado por el menú de acceso directo.
-

✍️ **Identificar el control asociado a un ContextMenu**

1. En la Lista de tareas, hacer doble clic en **TODO: use the SourceControl property**.
 2. Añadir las siguientes instrucciones de código debajo de la línea **TODO**:
- ```
Panel1.Controls.Eliminar(ContextMenu1.SourceControl)
```
3. Ejecutar la aplicación, hacer clic con el botón derecho en el control **Label** que muestra la ruta de archivo Chai.txt y, a continuación, hacer clic en **Delete from list**.
  4. En el menú **View**, hacer clic en **Show Previously Opened Files**, y abrir el menú **File**.
  5. Cerrar el menú **File** y cerrar la aplicación.
  6. ¿Qué método de un objeto **MenuItem** se utiliza para añadir un elemento de menú en tiempo de ejecución?

**El método Add.**

### ✎ Añadir un elemento de menú en tiempo de ejecución

1. En la Lista de tareas, hacer doble clic en **TODO: add a menu item to a menu**.
2. Añadir las siguientes instrucciones de código debajo de la línea **TODO:**  

```
fileMenuItem.MenuItems.Add(previousFile1)
```
3. Ejecutar la aplicación y, a continuación, en el menú **View**, hacer clic en **Show Previously Opened Files**.
4. Abrir el menú **File**, y abrir el menú **View**.  
Observar que el elemento se ha agregado al menú **File**.
5. Cerrar la aplicación.
6. ¿Existe algún modo para mostrar al usuario que actualmente estamos mostrando el archivo abierto anteriormente en el menú **File**?

**Podemos utilizar la propiedad *Checked* de un elemento de menú para indicar cuándo se ha seleccionado un elemento de menú. Activando y desactivando esta propiedad cada vez que se hace clic y añadiendo el código adecuado a nuestra aplicación, podemos hacer que el elemento de menú se comporte como un botón de opción.**

---

---

### ✎ Utilizar la propiedad *Checked* de un elemento de menú para señalar al usuario

1. En la Lista de tareas, hacer doble clic en **TODO: display a menu item as checked**.
2. Añadir las siguientes instrucciones de código debajo de la línea **TODO:**  

```
viewShowPreviousMenuItem.Checked = True
```
3. Ejecutar la aplicación y, a continuación, en el menú **View**, hacer clic en **Show Previously Opened Files**.
4. Abrir el menú **File**, y abrir el archivo **View**.
5. En el menú **View**, hacer clic en **Show Previously Opened Files**.
6. Abrir el archivo **File**, y abrir el menú **View**.
7. Si disponemos de tiempo, examinar el código utilizado para eliminar el elemento de menú del menú **File** y el código utilizado para responder a los eventos clic del segundo menú contextual.
8. Cerrar la aplicación, guardar los archivos del proyecto, y cerrar Visual Studio .NET.

## Lección: validar la entrada de los usuarios

- Cómo validar controles utilizando el evento **Validating**
- Control **ErrorProvider**
- Cómo utilizar el control **ErrorProvider**
- Demostración: validar datos en una aplicación Windows Forms

\*\*\*\*\*

### Introducción

Tanto si estamos escribiendo una sencilla aplicación calculadora como el interfaz para una compleja base de datos, necesitaremos validar información introducida por los usuarios. Esta lección describe cómo validar la entrada de los usuarios en una aplicación Windows Forms. También describe cómo mostrar un mensaje de error si la entrada del usuario no es válida.

### Objetivos de la lección

En esta lección, estudiaremos cómo:

- Validar la entrada de los usuarios utilizando el evento **Validating** de un control.
- Utilizar el control **ErrorProvider** para notificar al usuario cuándo un valor introducido no cumple los criterios de aceptación.
- Mostrar mensajes de error apropiados cuando los usuarios introducen datos no válidos.

## Cómo validar controles utilizando el evento Validating

- Utilizar el evento **Validating** de un control para validar la entrada de los usuarios
- [EjemploCódigo](#)
- El evento **Validated** se dispara cuando la validación de los controles finaliza la ejecución de eventos de validación
  - La propiedad **CausesValidation** determina si el control anterior participará en la validación. Si está establecida en **False** para un control, el control anterior no dispara el evento de validación

\*\*\*\*\*

### Introducción

En la mayoría de aplicaciones, un usuario introduce información que la aplicación procesa. La validación de datos garantiza que todos los valores de datos introducidos en nuestra aplicación son exactos y de un tipo válido.

Visual Studio .NET incluye el evento **Validating** para los controles, que se invoca antes de que un control pierda el foco. El evento únicamente se invoca cuando la propiedad **CausesValidation** del control que va a recibir el foco se establece en **True** (el valor predeterminado). Utilizar el evento **Validating** y la propiedad **CausesValidation** de un control para evaluar la entrada antes de permitir al usuario perder el foco de ese control.

### El evento Validating

La manera más sencilla de validar datos en una aplicación Windows Forms es utilizar el evento **Validating**. El evento **Validating** también permite controlar cuando puede cambiarse el foco a otros controles. Utilizando el evento **Validating**, podemos impedir que el foco cambie a otro control hasta que se hayan cumplido todas las normas de validación. Algunos usos posibles del evento **Validating** incluyen:

- Una aplicación de entrada de datos debe realizar una validación más sofisticada que la proporcionada por el control **Masked Edit**.
- Un formulario debe impedir que los usuarios abandonen un control, presionando TAB o una tecla de acceso directo, hasta que hayan introducido datos en un campo.

### La propiedad CausesValidation

La propiedad **CausesValidation** junto con el evento **Validating** limitan cuándo puede perder el foco un control. Podemos establecer la propiedad **CausesValidation** para determinar si el evento **Validating** ocurrirá en un segundo control desde el que se ha cambiado el foco. Si el evento **Validating** está siendo gestionado para un **TextBox**, y el usuario hace clic en un botón cuya propiedad **CausesValidation** está establecida en **True**, el evento **Validating** para el cuadro de texto se disparará. Sin embargo, si la propiedad **CausesValidation** para el botón se establece en **False**, el evento **Validating** no se disparará. De modo predeterminado, la propiedad **CausesValidation** está establecida en **True** para todos los controles.

**Procedimiento: validar un control**

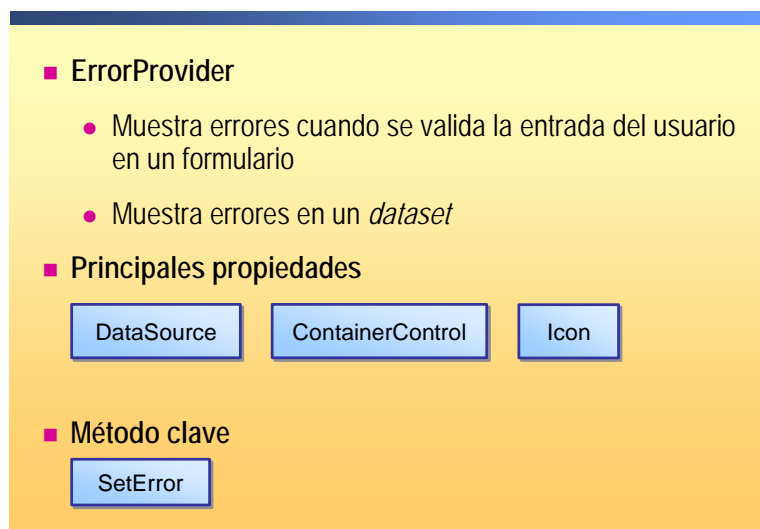
Para utilizar el evento **Validating** de un control **TextBox**:

1. Añadir un control **TextBox** a un formulario.
2. En el Editor de código, en la lista de procedimientos, hacer clic en el evento **Validating**.
3. Escribir un código de validación en el evento **Validating** para el **TextBox**.

```
Private Sub minValueTextBox_Validating(ByVal sender As
Object, ByVal e As System.ComponentModel.CancelEventArgs)
Handles minValueTextBox.Validating
 If CInt(minValueTextBox.Text) >= _
 CInt(maxValueTextBox.Text) Then
 e.Cancel = True
 MessageBox.Show("You must enter a minimum" & _
 "value that is less than the maximum value")
 End If
```

4. Establecer la propiedad **CausesValidation** en **False** para los controles que no deseamos que se dispare el evento **Validating**, como un botón **Help**.

## Control ErrorProvider



\*\*\*\*\*

### Introducción

El control **ErrorProvider** de Windows Forms permite mostrar un mensaje de error si los datos introducidos por el usuario no son válidos. Se utiliza con la validación de entrada del usuario en un formulario o con la presentación de errores en un *dataset*.

### Ventajas del uso de ErrorProvider

Un proveedor de errores es una alternativa mejor que mostrar un mensaje de error en un cuadro de mensaje. Si mostramos el mensaje de error en un cuadro de mensaje, el mensaje deja de estar visible si el usuario desecha el cuadro de mensaje. En cambio, el control **ErrorProvider** muestra un icono de error junto al control pertinente. Cuando el usuario posiciona el cursor del ratón sobre el icono de error, aparece un consejo que muestra la cadena del mensaje de error.

### Propiedades de ErrorProvider

Las principales propiedades del control **ErrorProvider** son:

#### ■ ContainerControl

Debemos establecer la propiedad **ContainerControl** al contenedor apropiado para que el control **ErrorProvider** muestre un icono de error en el formulario. Cuando añadimos el control en el diseñador, la propiedad **ContainerControl** se establece automáticamente al formulario contenedor. Si añadimos el control en el código, debemos establecerlo manualmente.

#### ■ DataSource

Cuando se establece la propiedad **DataSource**, el control **ErrorProvider** muestra mensajes de error para un *dataset*.

#### ■ Icon

La propiedad **Icon** puede establecerse en un icono de error personalizado en lugar del predeterminado.

### Métodos ErrorProvider

El método principal del control **ErrorProvider** es el método **SetError**, el cual especifica el control junto al que debería aparecer el icono de error, y el valor de la cadena del mensaje de error.



## Cómo utilizar el control ErrorProvider

Para utilizar el control ErrorProvider

- 1** Añadir controles al formulario
- 2** Añadir el control ErrorProvider
- 3** Añadir código al evento Validating del primer control

[EjemploCódigo](#)

\*\*\*\*\*

### Introducción

Podemos utilizar un control **ErrorProvider** de Windows Forms para mostrar un icono de error cuando el usuario introduce datos no válidos.

### Procedimiento


Para mostrar un icono de error utilizando el control **ErrorProvider**:

1. Añadir al formulario los controles que deben validarse.
2. Añadir el control **ErrorProvider**.
3. Seleccionar el primer control y añadir código al controlador del evento **Validating**.

El siguiente código analiza valores numéricos en un cuadro de texto. Si los datos no son válidos, se muestra un icono de error junto al cuadro de texto.

```
Private Sub TextBox1_Validating(ByVal sender As Object,
ByVal e As System.ComponentModel.CancelEventArgs) Handles
TextBox1.Validating
 'If the value in Textbox1 is not numeric
 If Not IsNumeric(TextBox1.Text) Then
 'Use the SetError method of the ErrorProvider
 ErrorProvider1.SetError(TextBox1, _
 "Not a numeric value.")
 'make sure the focus does not shift
 e.Cancel = True
 Else
 'If the value is numeric, set the error value to ""
 ErrorProvider1.SetError(TextBox1, "")
 End If
End Sub
```

## Demostración: validar datos en una aplicación Windows Forms



■ En esta demostración, veremos cómo

- Examinar las teclas pulsadas por el usuario
- Evitar que el foco cambie del control actual
- Utilizar un cuadro de mensaje para proporcionar sugerencias
- Utilizar un control **ErrorProvider** para proporcionar sugerencias
- Eliminar el icono **ErrorProvider** cuando el error deja de producirse
- Cambiar el icono que muestra **ErrorProvider**
- Permitir al usuario obtener ayuda

\*\*\*\*\*

### Introducción

En esta demostración, validaremos la entrada del usuario en una aplicación Windows Forms.

### Instrucciones

#### 🔍 Abrir el archivo **ValidatingInput.sln**

1. Abrir la solución **ValidatingInput.sln** en Visual Studio .NET desde la carpeta **demo03\Starter** que se puede encontrar dentro del fichero **demos07.zip**. La solución de este ejercicio se puede encontrar en la carpeta **demo03\Solution** del mismo fichero comprimido.
2. Abrir **Form1.vb** en la vista **Diseño**.
3. En el cuadro de herramientas, hacer doble clic en **ErrorProvider**.

#### 🔍 Examinar las teclas pulsadas por el usuario

1. Abrir **Form1.vb** en el Editor de código.
2. En el menú **Ver**, seleccionar **Mostrar tareas** y, a continuación, hacer clic en **Comentario**.
3. En la Lista de tareas, hacer doble clic en **TODO: check for valid characters**.
4. Añadir las siguientes instrucciones de código debajo de la línea **TODO**:

```
If e.KeyChar.IsDigit(e.KeyChar) = False Then
 e.Handled = True
 MessageBox.Show("Integer numbers only")
End If
```

5. Ejecutar la aplicación, presionar una tecla alfabética del teclado y, a continuación, hacer clic en **OK**.
6. Cerrar la aplicación.
7. ¿Qué ocurre si no se incluye la instrucción de código que establece el valor de la propiedad **Handled** en **True**?

**El carácter que representa la tecla que se ha pulsado se añadirá al cuadro de texto.**

### ⚡ Evitar que el control actual pierda el foco

1. En la Lista de tareas, hacer doble clic en **TODO: don't let the focus shift**.
2. Añadir la siguiente instrucción de código debajo de la línea TODO:  

```
e.Cancel = True
```
3. Ejecutar la aplicación, escribir **15** y hacer clic en **Submit Data and Exit**.
4. Escribir **5** y hacer clic en **Submit Data and Exit**.
5. Cerrar la aplicación.

### ⚡ Utilizar un cuadro de mensaje para proporcionar sugerencias

1. En la Lista de tareas, hacer doble clic en **TODO: give the user feedback using a message box**.
2. Añadir la siguiente instrucción de código debajo de la línea TODO:  

```
MessageBox.Show("You must enter a minimum value that " & _
"is less than the maximum value")
```
3. Ejecutar la aplicación, escribir **15** y hacer clic en **Submit Data and Exit**.
4. Escribir **5** y hacer clic en **Submit Data and Exit**.
5. Cerrar la aplicación.

### ⚡ Utilizar un control ErrorProvider para proporcionar sugerencias

1. Abrir Form1.vb en el Editor de código.
2. En la Lista de tareas, hacer doble clic en **TODO: use the error provider to provide a message**.
3. Añadir la siguiente instrucción de código debajo de la línea TODO:  

```
ErrorProvider1.SetError(epMinTextBox, _
"The minimum value must be smaller " & _
"than the maximum value")
```
4. Ejecutar la aplicación, presionar TAB dos veces, escribir **15** y hacer clic en **Submit Data and Exit**.
5. Posicionar el ratón sobre el icono **ErrorProvider** que se muestra a la derecha del cuadro de texto.
6. Escribir **5** y hacer clic en de uno de los otros cuadros de texto del formulario.
7. ¿Qué podemos hacer para eliminar el mensaje de ErrorProvider?

**Establecer el valor de la descripción de error de la propiedad SetError en longitud de cadena igual a cero.**

---

---

8. Cerrar la aplicación.

### ✍ Eliminar el icono **ErrorProvider** cuando el error deja de producirse

1. En la Lista de tareas, hacer doble clic en **TODO: reset the error provider**.
2. Añadir la siguiente instrucción de código debajo de la línea **TODO**:  

```
ErrorProvider1.SetError(epMinTextBox, "")
```
3. Ejecutar la aplicación, presionar TAB dos veces, escribir **15** y hacer clic en **Submit Data and Exit**.
4. Escribir **5** y hacer clic en uno de los otros cuadros de texto del formulario.
5. Cerrar la aplicación.

### ✍ Cambiar el icono que muestra **ErrorProvider**

1. En la Lista de tareas, hacer doble clic en **TODO: change the icon displayed by the error provider**.
2. Añadir las siguientes instrucciones de código debajo de la línea **TODO**:  

```
Dim ico As New Icon(Application.StartupPath & _
 "\msgbox01.ico")
ErrorProvider1.Icon = ico
```
3. Ejecutar la aplicación, presionar TAB dos veces, escribir **6** y presionar TAB.
4. Escribir **4** y hacer clic en **Submit Data and Exit**.
5. Escribir **8** y hacer clic en **Submit Data and Exit**.

### ✍ Permitir al usuario obtener ayuda

1. Ejecutar la aplicación, presione TAB, presione TAB de nuevo, escribir **15** y hacer clic en **Help**.
2. Escribir **4** y hacer clic en **Submit Data and Exit**.
3. Abrir Form1.vb en la vista Diseño.
4. Hacer clic en **Help** y, a continuación, en la ventana Propiedades, establecer el valor de **CausesValidation** en False.
5. Ejecutar la aplicación, presionar TAB dos veces, escribir **15** y hacer clic en **Help**.
6. Cerrar la aplicación.
7. Guardar el proyecto y cerrar Visual Studio .NET.