EBook Gratis

APRENDIZAJE winforms

Free unaffiliated eBook created from
Stack Overflow contributors.

#winforms

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: winforms

It is an unofficial and free winforms ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official winforms.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Primeros pasos con winforms

Observaciones

Windows Forms ("WinForms" para abreviar) es una biblioteca de clase GUI incluida con .NET Framework. Es un envoltorio sofisticado orientado a objetos alrededor de la API de Win32 , que permite el desarrollo de aplicaciones de escritorio y móviles de Windows orientadas a .NET Framework .

WinForms es principalmente impulsado por eventos . Una aplicación consta de varios *formularios* (mostrados como ventanas en la pantalla), que contienen *controles* (etiquetas, botones, cuadros de texto, listas, etc.) con los que el usuario interactúa directamente. En respuesta a la interacción del usuario, estos controles generan eventos que el programa puede manejar para realizar tareas.

Como en Windows, todo en WinForms es un control, que en sí mismo es un tipo de ventana. La clase de Control base proporciona una funcionalidad básica, que incluye propiedades para configurar texto, ubicación, tamaño y color, así como un conjunto común de eventos que se pueden manejar. Todos los controles se derivan de la clase Control, agregando características adicionales. Algunos controles pueden alojar otros controles, ya sea por reutilización (`Form` , `UserControl`) o diseño (`TableLayoutPanel` , `FlowLayoutPanel`).

WinForms ha sido compatible desde la versión original de .NET Framework (v1.0) y todavía está disponible en versiones modernas (v4.5). Sin embargo, ya no está en desarrollo activo y no se están agregando nuevas funciones. Según 9 desarrolladores de Microsoft en la conferencia Build 2014 :

> Windows Forms sigue siendo compatible, pero en modo de mantenimiento. Arreglarán los errores a medida que se descubran, pero la nueva funcionalidad está fuera de la tabla.

La biblioteca Mono multiplataforma y de código abierto proporciona una implementación básica de Windows Forms, que admite todas las características que la implementación de Microsoft hizo a partir de .NET 2.0. Sin embargo, WinForms no se desarrolla activamente en Mono y una implementación completa se considera imposible, dado que el marco está inextricablemente vinculado con la API de Windows nativa (que no está disponible en otras plataformas).

Ver también:

- Documentación de formularios Windows en MSDN

Examples

Creando una aplicación simple de WinForms usando Visual Studio

Este ejemplo le mostrará cómo crear un proyecto de aplicación de Windows Forms en Visual

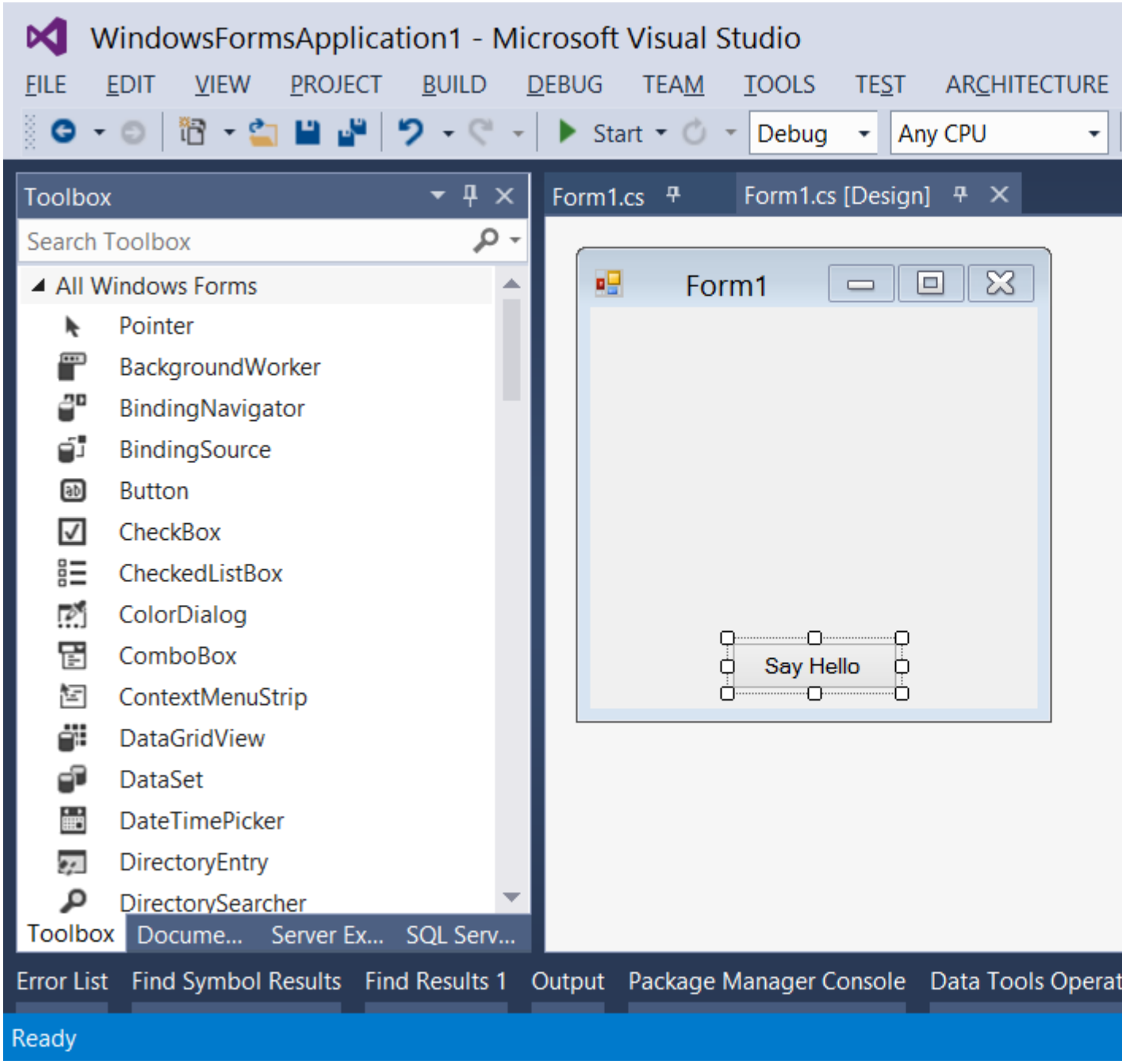
Studio.

Crear proyecto Windows Forms

1. Inicie Visual Studio.
2. En el menú **Archivo** , seleccione **Nuevo** y, a continuación, seleccione **Proyecto** . Aparece el cuadro de diálogo **Nuevo proyecto** .
3. En el panel **Plantillas instaladas** , seleccione "Visual C #" o "Visual Basic".
4. Sobre el panel central, puede seleccionar el marco de destino de la lista desplegable.
5. En el panel central, seleccione la plantilla de **aplicación de Windows Forms** .
6. En el cuadro de texto **Nombre** , escriba un nombre para el proyecto.
7. En el cuadro de texto **Ubicación** , elija una carpeta para guardar el proyecto.
8. Haga clic en **Aceptar** .
9. El Diseñador de Windows Forms se abre y muestra **Form1** del proyecto.

Añadir controles al formulario

1. Desde la paleta de la **caja de herramientas** , arrastre un control **Button** al formulario.
2. Haga clic en el botón para seleccionarlo. En la ventana Propiedades, establezca la propiedad `Text` en **Saludar** .



Escribir código

1. Haga doble clic en el botón para agregar un controlador de eventos para el evento `Click` . El Editor de código se abrirá con el punto de inserción ubicado dentro de la función del controlador de eventos.

2. Escriba el siguiente código:

   DO#

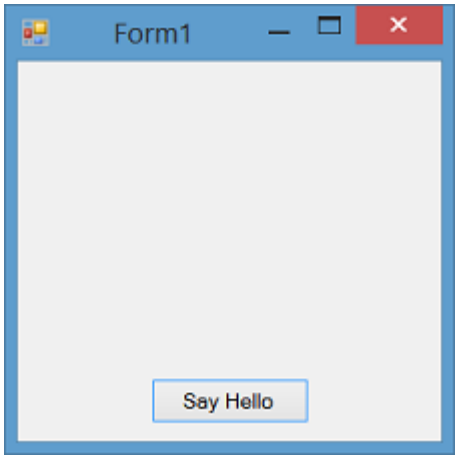
   ```
   MessageBox.Show("Hello, World!");
   ```

   VB.NET

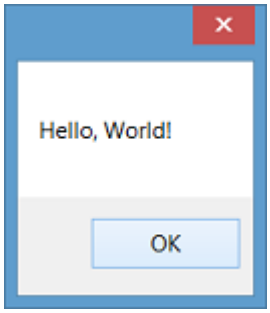
```
MessageBox.Show("Hello, World!")
```

Correr y probar

1. Presione `F5` para ejecutar la aplicación.



2. Cuando su aplicación se esté ejecutando, haga clic en el botón para ver "¡Hola, mundo!" mensaje.



3. Cierre el formulario para volver a Visual Studio.

Creando una aplicación simple de WinForms C # usando un editor de texto

1. Abra un editor de texto (como el Bloc de notas) y escriba el código a continuación:

```
using System;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;

namespace SampleApp
{
    public class MainForm : Form
    {
        private Button btnHello;

        // The form's constructor: this initializes the form and its controls.
        public MainForm()
        {
            // Set the form's caption, which will appear in the title bar.
```

```
            this.Text = "MainForm";

            // Create a button control and set its properties.
            btnHello = new Button();
            btnHello.Location = new Point(89, 12);
            btnHello.Name = "btnHello";
            btnHello.Size = new Size(105, 30);
            btnHello.Text = "Say Hello";

            // Wire up an event handler to the button's "Click" event
            // (see the code in the btnHello_Click function below).
            btnHello.Click += new EventHandler(btnHello_Click);

            // Add the button to the form's control collection,
            // so that it will appear on the form.
            this.Controls.Add(btnHello);
        }

        // When the button is clicked, display a message.
        private void btnHello_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Hello, World!");
        }

        // This is the main entry point for the application.
        // All C# applications have one and only one of these methods.
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.Run(new MainForm());
        }
    }
}
```

2. Guarde el archivo en una ruta a la que tenga acceso de lectura / escritura. Es convencional `X:\MainForm.cs` nombre al archivo después de la clase que contiene, por ejemplo, `X:\MainForm.cs` .

3. Ejecute el compilador de C # desde la línea de comandos, pasando la ruta al archivo de código como un argumento:

   ```
   %WINDIR%\Microsoft.NET\Framework64\v4.0.30319\csc.exe /target:winexe "X:\MainForm.cs"
   ```

   Nota: Para usar una versión del compilador de C # para otras versiones de .NET framework, eche un vistazo a la ruta, `%WINDIR%\Microsoft.NET` y modifique el ejemplo anterior según corresponda. Para obtener más información sobre la compilación de aplicaciones de C #, consulte Compilar y ejecutar su primer programa de C # .

4. Una vez que se haya completado la compilación, se `MainForm.exe` una aplicación llamada `MainForm.exe` en el mismo directorio que el archivo de código. Puede ejecutar esta aplicación desde la línea de comandos o haciendo doble clic en ella en el Explorador.

Creando una aplicación simple de WinForms VB.NET usando un editor de

texto

1. Abra un editor de texto (como el Bloc de notas) y escriba el código a continuación:

```
Imports System.ComponentModel
Imports System.Drawing
Imports System.Windows.Forms

Namespace SampleApp
    Public Class MainForm : Inherits Form
        Private btnHello As Button

        ' The form's constructor: this initializes the form and its controls.
        Public Sub New()
            ' Set the form's caption, which will appear in the title bar.
            Me.Text = "MainForm"

            ' Create a button control and set its properties.
            btnHello = New Button()
            btnHello.Location = New Point(89, 12)
            btnHello.Name = "btnHello"
            btnHello.Size = New Size(105, 30)
            btnHello.Text = "Say Hello"

            ' Wire up an event handler to the button's "Click" event
            ' (see the code in the btnHello_Click function below).
            AddHandler btnHello.Click, New EventHandler(AddressOf btnHello_Click)

            ' Add the button to the form's control collection,
            ' so that it will appear on the form.
            Me.Controls.Add(btnHello)
        End Sub

        ' When the button is clicked, display a message.
        Private Sub btnHello_Click(sender As Object, e As EventArgs)
            MessageBox.Show("Hello, World!")
        End Sub

        ' This is the main entry point for the application.
        ' All VB.NET applications have one and only one of these methods.
        <STAThread> _
        Public Shared Sub Main()
            Application.EnableVisualStyles()
            Application.Run(New MainForm())
        End Sub
    End Class
End Namespace
```

2. Guarde el archivo en una ruta a la que tenga acceso de lectura / escritura. Es convencional nombrar el archivo después de la clase que contiene, por ejemplo, `X:\MainForm.vb` .

3. Ejecute el compilador VB.NET desde la línea de comandos, pasando la ruta al archivo de código como un argumento:

```
%WINDIR%\Microsoft.NET\Framework64\v4.0.30319\vbc.exe /target:winexe "X:\MainForm.vb"
```

Nota: Para usar una versión del compilador VB.NET para otras versiones de .NET

framework, eche un vistazo a la ruta `%WINDIR%\Microsoft.NET` y modifique el ejemplo anterior en consecuencia. Para obtener más información sobre la compilación de aplicaciones VB.NET, consulte Hello World .

4. Una vez que se haya completado la compilación, se `MainForm.exe` una aplicación llamada `MainForm.exe` en el mismo directorio que el archivo de código. Puede ejecutar esta aplicación desde la línea de comandos o haciendo doble clic en ella en el Explorador.

Lea Primeros pasos con winforms en línea:
https://riptutorial.com/es/winforms/topic/1018/primeros-pasos-con-winforms

Capítulo 2: Caja de texto

Examples

Autocompletar a partir de una colección de cadenas

```
var source = new AutoCompleteStringCollection();

// Add your collection of strings.
source.AddRange(new[] { "Guybrush Threepwood", "LeChuck" });

var textBox = new TextBox
{
    AutoCompleteCustomSource = source,
    AutoCompleteMode = AutoCompleteMode.SuggestAppend,
    AutoCompleteSource = AutoCompleteSource.CustomSource
};

form.Controls.Add(textBox);
```

Esto se completará **automáticamente** cuando el usuario intente escribir **G** o **L.**

`AutoCompleteMode.SuggestAppend` Se mostrará tanto una lista de valores sugeridos y se auto escriba el primer partido, `Append` solamente y `Suggest` solamente están disponibles, también.

Permitir sólo dígitos en el texto.

```
textBox.KeyPress += (sender, e) => e.Handled = !char.IsControl(e.KeyChar) &&
!char.IsDigit(e.KeyChar);
```

Esto solo permitirá el uso de dígitos y caracteres de control en el `TextBox` , otras combinaciones son posibles utilizando el mismo método de configuración de la propiedad `Handle` en true para bloquear el texto.

El usuario todavía puede copiar / pegar caracteres no deseados, por lo que debería haber una verificación adicional en `TextChanged` para limpiar la entrada:

```
textBox.TextChanged += (sender, e) => textBox.Text = Regex.Match(textBox.Text, @"\d+").Value
```

En este ejemplo, una **expresión regular** se utiliza para filtrar el texto.

NumericUpDown debería preferirse a los números cuando sea posible.

Cómo desplazarse hasta el final.

```
textBox.SelectionStart = textBox.TextLength;
textBox.ScrollToCaret();
```

Aplicando el mismo principio, `SelectionStart` se puede establecer en `0` para desplazarse a la parte superior o a un número específico para ir a un carácter específico.

Agregar un marcador de posición al cuadro de texto

Este código coloca el texto de la *pista* en el formulario de carga y lo manipula de la siguiente manera:

DO#

```
private void Form_load(object sender, EventArgs e)
{
    textBox.Text = "Place Holder text...";
}

private void textBox_Enter(object sender, EventArgs e)
{
    if(textBox.Text == "Place Holder text...")
    {
        textBox.Text = "";
    }
}

private void textBox_Leave(object sender, EventArgs e)
{
    if(textBox.Text.Trim() == "")
    {
        textBox.Text = "Place Holder text...";
    }
}
```

VB.NET

```
Private Sub Form_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    textBox.Text = "Place Holder text..."
End Sub

Private Sub textBox_GotFocus(sender as Object,e as EventArgs) Handles textBox.GotFocus
    if Trim(textBox.Text) = "Place Holder text..." Then
        textBox.Text = ""
    End If
End Sub

Private Sub textBox_LostFocus(sender as Object,e as EventArgs) Handles textBox.LostFocus
    if Trim(textBox.Text) = "" Then
        textBox.Text = "Place Holder text..."
    End If
End Sub
```

Lea Caja de texto en línea: https://riptutorial.com/es/winforms/topic/4674/caja-de-texto

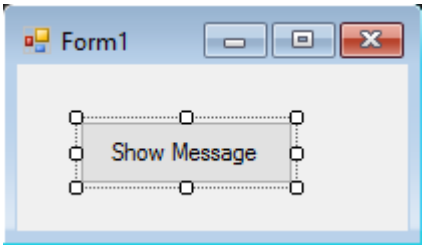
Capítulo 3: Controles basicos

Examples

Botón

Los botones son uno de los controles más simples y se usan principalmente para ejecutar algún código cuando el usuario lo desea.

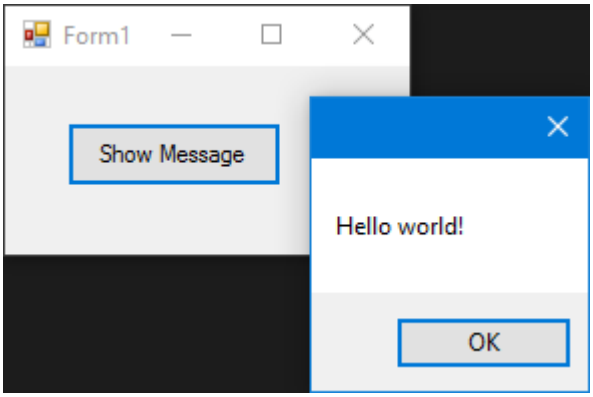
Aquí tenemos un caso muy simple, mostrar un cuadro de mensaje cuando se hace clic en un botón. `cmdShowMessage` un botón a un formulario, lo denominamos `cmdShowMessage` como se usa en el código para identificar el objeto y establecer el texto de los botones en Mostrar mensaje.



Solo necesitamos hacer doble clic en el botón del diseñador visual y Visual Studio generará el código para el evento de clic. Ahora solo necesitamos agregar el código para el MessageBox allí:

```
private void cmdShowMessage_Click(object sender, EventArgs e)
{
  MessageBox.Show("Hello world!");
}
```

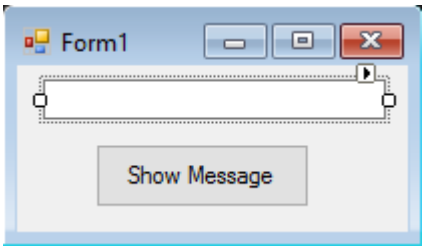
Si ejecutamos el programa ahora y hacemos clic en el botón, veremos el mensaje que aparece:



Caja de texto

Los cuadros de texto permiten al usuario ingresar datos en el programa.

Vamos a modificar el formulario y agregar un cuadro de texto para que el cuadro de mensaje nos muestre el mensaje que el usuario desea. Ahora nuestra forma se ve como:

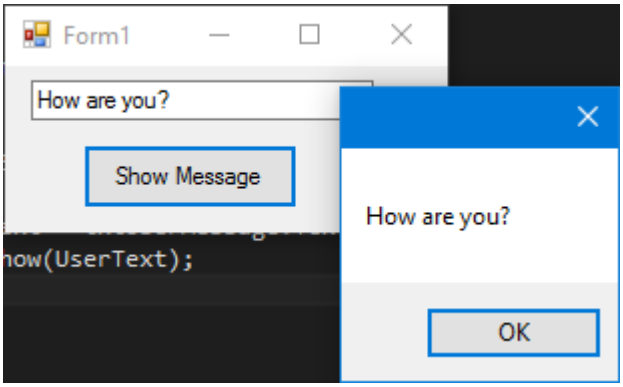


Y luego modifique el evento de clic de botón para usar el texto del cuadro de texto:

```
private void cmdShowMessage_Click(object sender, EventArgs e)
{
  string UserText = txtUserMessage.Text;
  MessageBox.Show(UserText);
}
```

Como puede ver, estamos utilizando la propiedad `.Text` del `.Text` de texto que es el texto contenido en el cuadro de texto.

Si ejecutamos el programa, podremos escribir en el cuadro de texto. Cuando hagamos clic en el botón, el MessageBox mostrará el texto que hemos escrito:

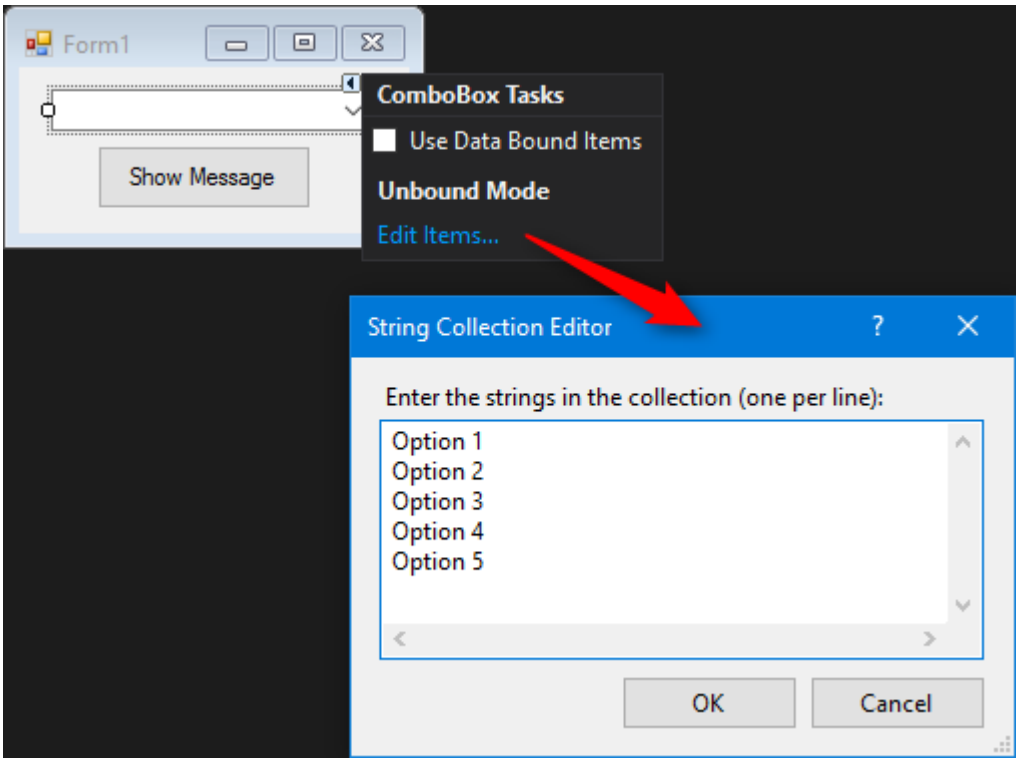


Caja combo

Los ComboBoxes permiten al usuario elegir una de varias opciones proporcionadas por el desarrollador.

Vamos a modificar el formulario y agregar un cuadro combinado para que el cuadro de mensaje nos muestre el mensaje que el usuario desea de una lista que le proporcionaremos.

Después de agregar el combo al formulario, ahora agregamos una lista de opciones al combo. Para ello necesitamos modificar la propiedad `Items` :

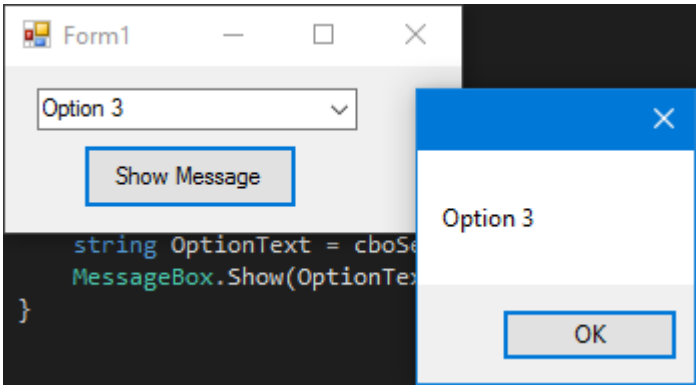


Ahora necesitamos modificar el código del evento click:

```
private void cmdShowMessage_Click(object sender, EventArgs e)
{
  string OptionText = cboSelectOption.SelectedItem.ToString();
  MessageBox.Show(OptionText);
}
```

Como puede ver, usamos la propiedad `SelectedItem` , que contiene el objeto de la opción seleccionada. Como necesitamos una cadena para mostrar y el compilador no sabe si el objeto es o no una cadena, necesitamos usar el método `ToString()` .

Si ejecutamos el programa, podremos elegir la opción que preferimos y cuando hagamos clic en el botón, el cuadro de mensaje lo mostrará:



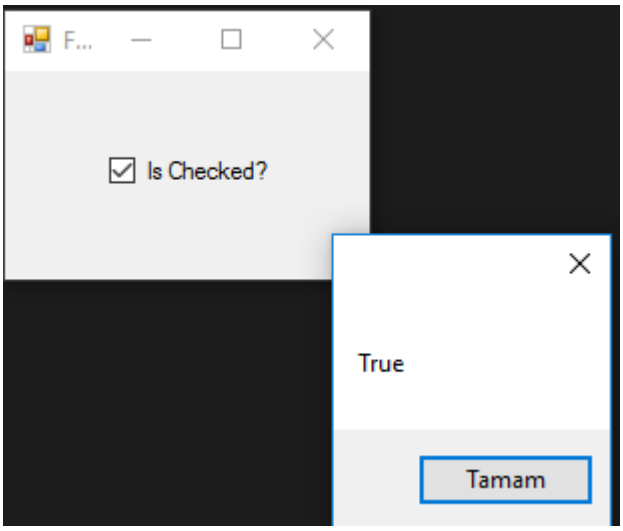
Para recibir una notificación cuando un usuario selecciona un elemento del cuadro combinado, use el evento `SelectionChangeCommitted` . Podríamos usar el evento `SelectedIndexChanged` , pero esto también se produce cuando cambiamos programáticamente el elemento seleccionado en el cuadro combinado.

Caja

Checkbox es un control que permite al usuario obtener valores `boolean` de un usuario para una pregunta específica como **"¿Estás bien?"** .

Tiene un evento llamado `CheckedChanged` , que ocurre cada vez que se cambia la propiedad de `check` .

Aquí hay un CheckBox que tiene una pregunta **"¿Se comprueba?"** .



Conseguimos este `MessageBox` del evento `CheckedChanged` ,

```
private void checkBox1_CheckedChanged(object sender, EventArgs e)
        {
            bool IsChecked = checkBox1.Checked;
            MessageBox.Show(IsChecked.ToString());
        }
```

Si **CheckBox** está marcado -> `IsChecked` variable `IsChecked` será `true` .

Si **CheckBox** no está marcado -> `IsChecked` variable `IsChecked` será `false` .

Cuadro de lista

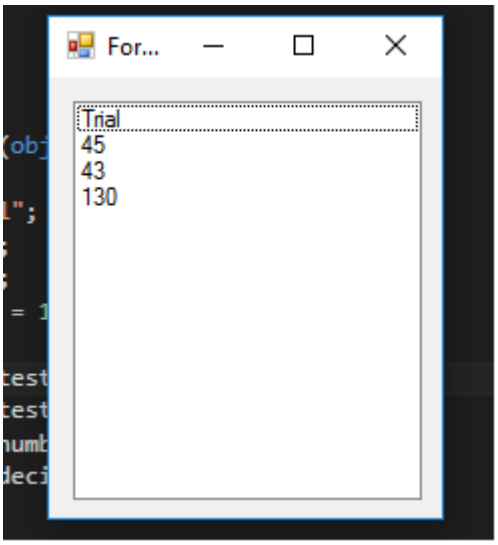
`Listbox` es un control que puede contener colección de objetos. `Listbox` es similar a `Combobox` pero en `Combobox` ; Sólo los elementos seleccionados son visibles para el usuario. En `Listbox` ; todos los elementos son visibles para el usuario.

¿Cómo agregar elementos a ListBox?

```
private void Form3_Load(object sender, EventArgs e)
        {
            string test = "Trial";
            string test2 = "45";
            int numberTest = 43;
            decimal decimalTest = 130;
```

```
            listBox1.Items.Add(test);
            listBox1.Items.Add(test2);
            listBox1.Items.Add(numberTest);
            listBox1.Items.Add(decimalTest);
        }
```

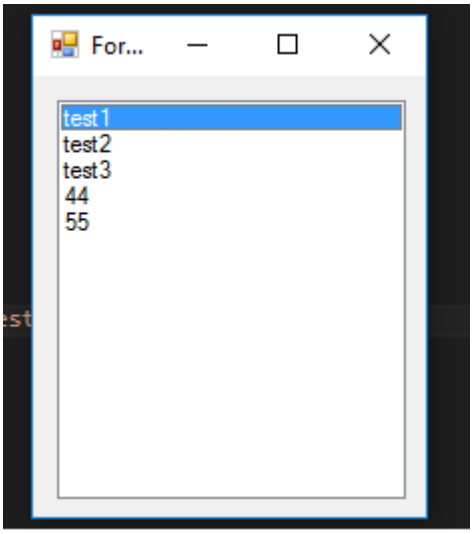
Salida ;



O se pueden dar `datasources` ,

```
private void Form3_Load(object sender, EventArgs e)
        {
            List<string> TestList = new List<string> { "test1", "test2", "test3", "44", "55"
};
            listBox1.DataSource = TestList;
        }
```

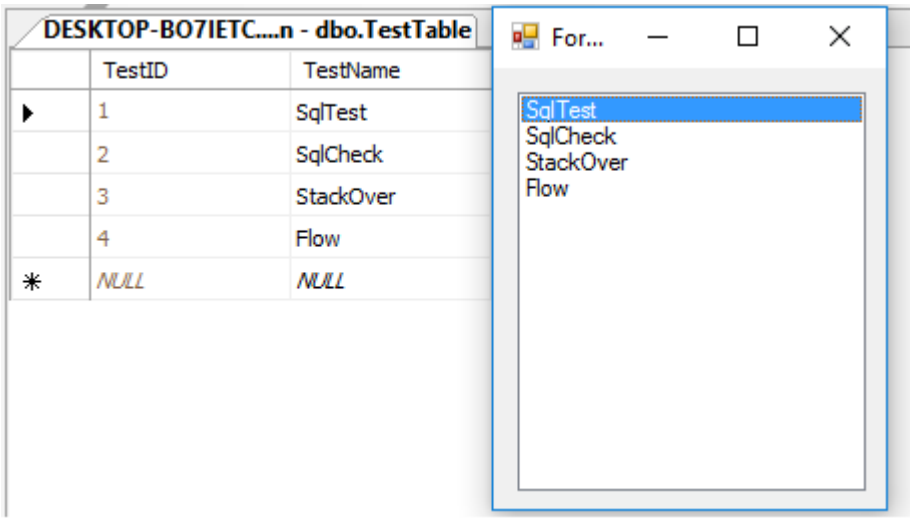
Salida;




```
private void Form3_Load(object sender, EventArgs e)
        {
            SqlConnection Connection = new
SqlConnection("Server=serverName;Database=db;Trusted_Connection=True;"); //Connetion to MS-
```

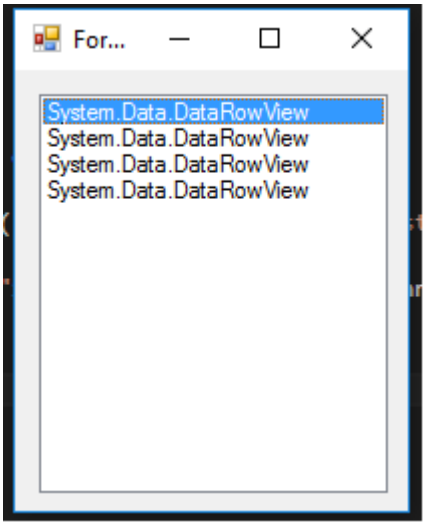
```
SQL(RDBMS)
            Connection.Open(); //Connection open
            SqlDataAdapter Adapter = new SqlDataAdapter("Select * From TestTable",
Connection); // Get all records from TestTable.
            DataTable DT = new DataTable();
            Adapter.Fill(DT); // Fill records to DataTable.
            listBox1.DataSource = DT; // DataTable is the datasource.
            listBox1.ValueMember = "TestID";
            listBox1.DisplayMember= "TestName";
        }
```

La **salida adecuada** ;



Dar un origen de datos sql externo al cuadro de lista requiere, `ValueMember` y `DisplayMember`

Si **NO** se verá así,



Eventos útiles;

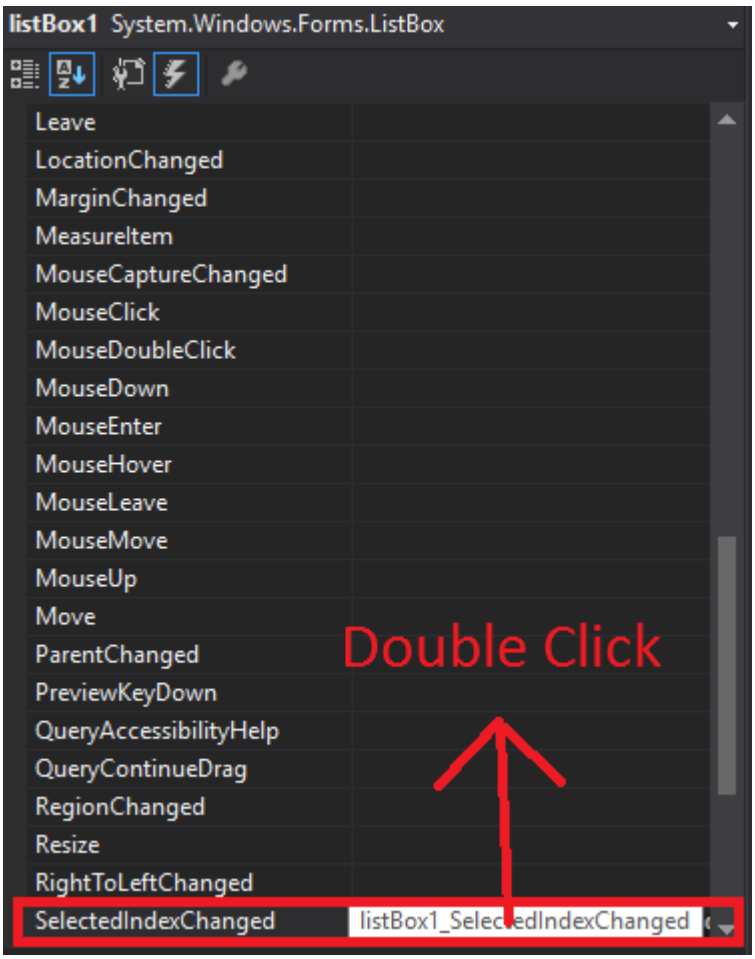
SelectedIndex_Changed;

Definir una lista para dar fuente de datos

```
private void Form3_Load(object sender, EventArgs e)
```

```
        {
            List<string> DataList = new List<string> {"test1" , "test2" , "test3" , "44" ,
"45" };
            listBox1.DataSource = TestList;
        }
```

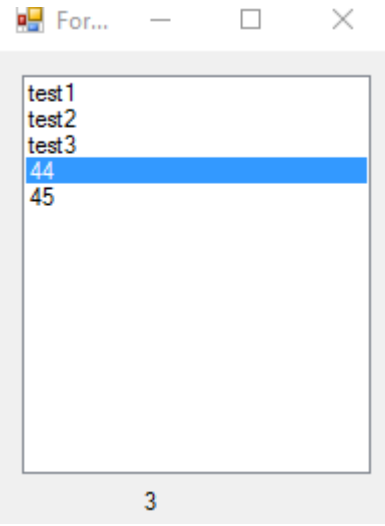
En el diseño del formulario, seleccione `Listbox` y presione F4 o, a la derecha, haga clic en el icono iluminado.



Visual Studio generará `listBox1_SelectedIndexChanged` a codebehind.

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
            int Index = listBox1.SelectedIndex;
            label1.Text = Index.ToString();
        }
```

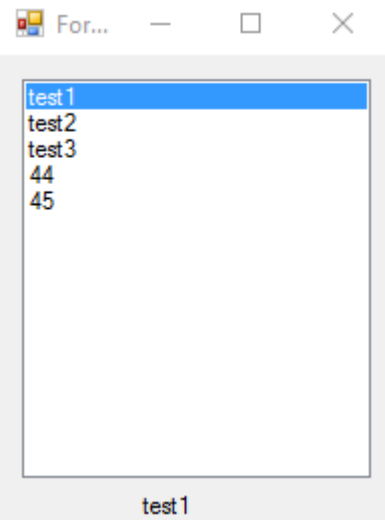
Resultado de `SelectedIndex_Changed` ; (la etiqueta en la parte inferior mostrará el índice de cada elemento seleccionado)



SelectedValue_Changed; (El origen de datos es el mismo que en la parte superior y puede generar este evento como SelectedIndex_Changed)

```
private void listBox1_SelectedValueChanged(object sender, EventArgs e)
        {
            label1.Text = listBox1.SelectedValue.ToString();
        }
```

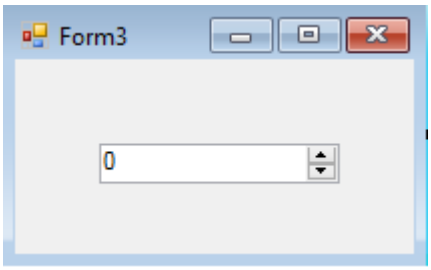
Salida ;



NumericUpDown

NumericUpDown es un control que se parece a TextBox. Este control permite al usuario mostrar / seleccionar un número desde un rango. Las flechas arriba y abajo están actualizando el valor del cuadro de texto.

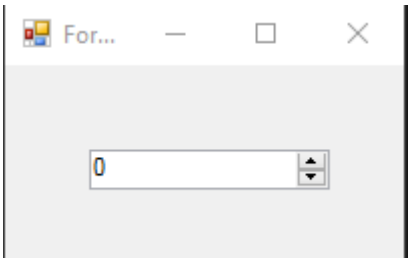
El control se ve como



En `Form_Load` se puede establecer el rango.

```
private void Form3_Load(object sender, EventArgs e)
    {
        numericUpDown1.Maximum = 10;
        numericUpDown1.Minimum = -10;
    }
```

Salida;

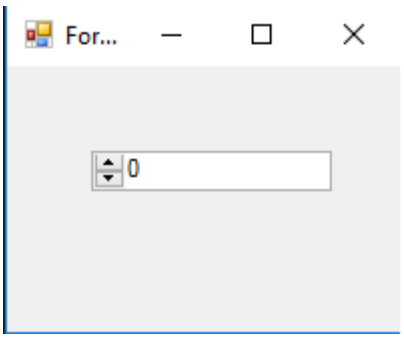


UpDownAlign establecerá la posición de las flechas;

```
 private void Form3_Load(object sender, EventArgs e)
        {
            numericUpDown1.UpDownAlign = LeftRightAlignment.Left;

        }
```

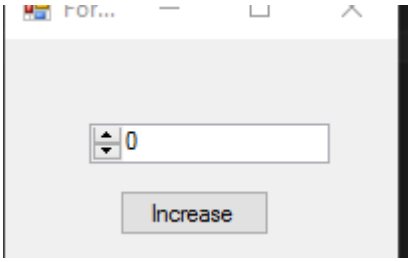
Salida;



Método `UpButton()` aumenta el número del control. (Se puede llamar desde cualquier lugar. Usé un `button` para llamarlo).

```
private void button1_Click(object sender, EventArgs e)
        {
            numericUpDown1.UpButton();
        }
```

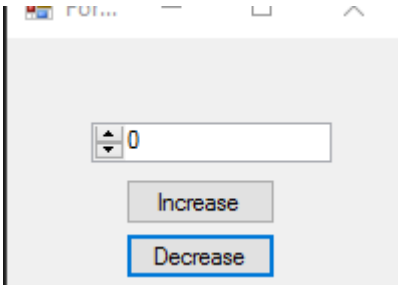
**Salida



`DownButton()` Método disminuye el número del control. (Se puede llamar desde cualquier lugar. Usé un `button` para volver a llamar.)

```
 private void button2_Click(object sender, EventArgs e)
        {
            numericUpDown1.DownButton();
        }
```

Salida;



Eventos utiles

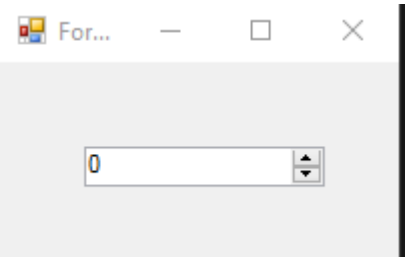
ValueChanged;

Ese evento funcionará cuando haga clic en la flecha hacia arriba o hacia abajo.

```
private void numericUpDown1_ValueChanged(object sender, EventArgs e)
        {
            decimal result = numericUpDown1.Value; // it will get the current value
            if (result == numericUpDown1.Maximum) // if value equals Maximum value that we set
in Form_Load.
            {
                label1.Text = result.ToString() + " MAX!"; // it will add "MAX" at the end of
the label
            }
            else if (result == numericUpDown1.Minimum) //  if value equals Minimum value that
we set in Form_Load.
            {
                label1.Text = result.ToString() + " MIN!"; // it will add "MIN" at the end of
the label
            }
            else
            {
                label1.Text = result.ToString(); // If Not Max or Min, it will show only the
number.
```

```
            }
        }
```

Salida ;



Lea Controles basicos en línea: https://riptutorial.com/es/winforms/topic/5816/controles-basicos

Capítulo 4: Controles hereditarios

Observaciones

Los controles se derivan exactamente de la misma manera que otras clases. Lo único de lo que hay que tener cuidado es anular eventos: por lo general, es recomendable asegurarse de llamar al controlador de eventos base después del suyo. Mi propia regla de oro: en caso de duda, llame al evento base.

Examples

Ajustes de toda la aplicación

Una lectura rápida de la mayoría de los sitios de desarrolladores revelará que WinForms se considera inferior a WPF. Una de las razones más citadas es la supuesta dificultad para realizar cambios amplios en la aplicación del "look-and-feel" de una aplicación completa.

De hecho, es sorprendentemente fácil producir una aplicación en WinForms que sea fácilmente configurable tanto en tiempo de diseño como en tiempo de ejecución, si simplemente evita el uso de los controles estándar y deriva los suyos propios.

Tomemos el TextBox como ejemplo. Es difícil imaginar una aplicación de Windows que no requiera el uso de un TextBox en algún momento u otro. Por lo tanto, tener tu propio TextBox siempre tendrá sentido. Tomemos el siguiente ejemplo:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace StackOverflowDocumentation
{
    public class SOTextBox : TextBox
    {
        public SOTextBox() : base()
        {
            base.BackColor = SOUserPreferences.BackColor;
            base.ForeColor = SOUserPreferences.ForeColor;
        }
        protected override void OnEnter(EventArgs e)
        {
            base.BackColor = SOUserPreferences.FocusColor;
            base.OnEnter(e);
        }
        protected override void OnLeave(EventArgs e)
        {
            base.BackColor = SOUserPreferences.BackColor;
            base.OnLeave(e);
        }
    }
```

```
}
```

Una de las cosas que los usuarios encuentran más útil en un formulario de ingreso de datos, con muchos cuadros de entrada, es tener el color de fondo del cuadro con cambio de enfoque. Visiblemente es más fácil de ver que un cursor vertical estándar parpadeante. El código anterior proporciona un TextBox que hace precisamente eso.

En el proceso hace uso de las propiedades estáticas de una clase estática. Doy a continuación un extracto de la mía:

```
using System;
using System.Threading;
using Microsoft.Win32;
using System.Globalization;
using System.Data;
using System.Drawing;

namespace StackOverflowDocumentation
{
    public class SOUserPreferences
    {
        private static string language;
        private static string logPath;
        private static int formBackCol;
        private static int formForeCol;
        private static int backCol;
        private static int foreCol;
        private static int focusCol;

        static SOUserPreferences()
        {
            try
            {
                RegistryKey HKCU = Registry.CurrentUser;
                RegistryKey kSOPrefs = HKCU.OpenSubKey("SOPrefs");
                if (kSOPrefs != null)
                {
                    language = kSOPrefs.GetValue("Language", "EN").ToString();
                    logPath = kSOPrefs.GetValue("LogPath", "c:\\windows\\logs\\").ToString();
                    formForeCol = int.Parse(kSOPrefs.GetValue("FormForeColor", "-
2147483630").ToString());
                    formBackCol = int.Parse(kSOPrefs.GetValue("FormBackColor", "-
2147483633").ToString());
                    foreCol = int.Parse(kSOPrefs.GetValue("ForeColor", "-
2147483640").ToString());
                    backCol = int.Parse(kSOPrefs.GetValue("BackColor", "-
2147483643").ToString());
                    focusCol = int.Parse(kSOPrefs.GetValue("FocusColor", "-
2147483643").ToString());
                }
                else
                {
                    language = "EN";
                    logPath = "c:\\windows\\logs\\";
                    formForeCol = -2147483630;
                    formBackCol = -2147483633;
                    foreCol = -2147483640;
                    backCol = -2147483643;
```

```
                focusCol = -2147483643;
            }
        }
        catch (Exception ex)
        {
            //handle exception here;
        }
    }

    public static string Language
    {
        get
        {
            return language;
        }
        set
        {
            language = value;
        }
    }

    public static string LogPath
    {
        get
        {
            return logPath;
        }
        set
        {
            logPath = value;
        }
    }

    public static Color FormBackColor
    {
        get
        {
            return ColorTranslator.FromOle(formBackCol);
        }
        set
        {
            formBackCol = ColorTranslator.ToOle(value);
        }
    }

    public static Color FormForeColor
    {
        get
        {
            return ColorTranslator.FromOle(formForeCol);
        }
        set
        {
            formForeCol = ColorTranslator.ToOle(value);
        }
    }

    public static Color BackColor
    {
        get
        {
```

```
                return ColorTranslator.FromOle(backCol);
            }
            set
            {
                backCol = ColorTranslator.ToOle(value);
            }
        }

        public static Color ForeColor
        {
            get
            {
                return ColorTranslator.FromOle(foreCol);
            }
            set
            {
                foreCol = ColorTranslator.ToOle(value);
            }
        }

        public static Color FocusColor
        {
            get
            {
                return ColorTranslator.FromOle(focusCol);
            }
            set
            {
                focusCol = ColorTranslator.ToOle(value);
            }
        }
    }
}
```

Esta clase utiliza el registro de Windows para conservar las propiedades, pero puede usar una base de datos o un archivo de configuración si lo prefiere. La ventaja de usar una clase estática de esta manera es que los cambios en toda la aplicación pueden realizarse no solo en tiempo de diseño, sino también por parte del usuario en tiempo de ejecución. Siempre incluyo un formulario en mis aplicaciones que permite al usuario cambiar los valores preferidos. La función de guardar no solo se guarda en el Registro (o base de datos, etc.), sino que también en tiempo de ejecución cambia las propiedades de la clase estática. Tenga en cuenta que las propiedades estáticas de una clase estática no son constantes; en este sentido pueden considerarse como variables de aplicación amplia. Esto significa que cualquier forma abierta después de los cambios que se guardan se verá afectada inmediatamente por los cambios guardados.

Podrá pensar fácilmente en otras propiedades de la aplicación que le gustaría que sean configurables de la misma manera. Las fuentes son otro muy buen ejemplo.

NumberBox

A menudo, deseará tener un cuadro de entrada que solo lleve números. De nuevo, derivando de los controles estándar, esto se logra fácilmente, por ejemplo:

```
using System;
using System.Windows.Forms;
```

```
using System.Globalization;

namespace StackOverflowDocumentation
{
    public class SONumberBox : SOTextBox
    {
        private int decPlaces;
        private int extraDecPlaces;
        private bool perCent;
        private bool useThouSep = true;
        private string decSep = ".";
        private string thouSep = ",";
        private double numVal;

        public SONumberBox() : base()

    {
    }

    public bool PerCent
    {
        get
        {
            return perCent;
        }
        set
        {
            perCent = value;
        }
    }

    public double Value
    {
        get
        {
            return numVal;
        }
        set
        {
            numVal = value;
            if (perCent)
            {
                double test = numVal * 100.0;
                this.Text = FormatNumber(test) + "%";
            }
            else
            {
                this.Text = FormatNumber(value);
            }
        }
    }
    public bool UseThousandSeparator
    {
        get
        {
            return useThouSep;
        }
        set
        {
            useThouSep = value;
        }
```

```
    }
    public int DecimalPlaces
    {
        get
        {
            return decPlaces;
        }
        set
        {
            decPlaces = value;
        }
    }
    public int ExtraDecimalPlaces
    {
        get
        {
            return extraDecPlaces;
        }
        set
        {
            extraDecPlaces = value;
        }
    }
    protected override void OnTextChanged(EventArgs e)
    {
        string newVal = this.Text;
        int len = newVal.Length;
        if (len == 0)
        {
            return;
        }
        bool neg = false;
        if (len > 1)
        {
            if (newVal.Substring(0, 1) == "-")
            {
                newVal = newVal.Substring(1, len - 1);
                len = newVal.Length;
                neg = true;
            }
        }
        double val = 1.0;
        string endChar = newVal.Substring(newVal.Length - 1);
        switch (endChar)
        {
            case "M":
            case "m":
                if (len > 1)
                {
                    val = double.Parse(newVal.Substring(0, len - 1)) * 1000000.0;
                }
                else
                {
                    val *= 1000000.0;
                }
                if (neg)
                {
                    val = -val;
                }
                this.Text = FormatNumber(val);
                break;
```

```
            case "T":
            case "t":
                if (len > 1)
                {
                    val = double.Parse(newVal.Substring(0, len - 1)) * 1000.0;
                }
                else
                {
                    val *= 1000.0;
                }
                if (neg)
                {
                    val = -val;
                }
                this.Text = FormatNumber(val);
                break;
        }

        base.OnTextChanged(e);
    }
    protected override void OnKeyPress(KeyPressEventArgs e)
    {
        bool handled = false;
        switch (e.KeyChar)
        {
            case '-':
                if (this.Text.Length == 0)
                {
                    break;
                }
                else if (this.SelectionStart == 0)
                {
                    //negative being inserted first
                    break;
                }
                else
                {
                    handled = true;
                    break;
                }
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9':
            case '0':
            case (char)Keys.Back:
                break;
            case 'M':
            case 'm':
            case 'T':
            case 't':
            case '%':
                //check last pos
                int l = this.Text.Length;
                int sT = this.SelectionStart;
                int sL = this.SelectionLength;
```

```
                if ((sT + sL) != l)
                {
                    handled = true;
                }
                break;
            default:
                string thisChar = e.KeyChar.ToString();
                if (thisChar == decSep)
                {
                    char[] txt = this.Text.ToCharArray();
                    for (int i = 0; i < txt.Length; i++)
                    {
                        if (txt[i].ToString() == decSep)
                        {
                            handled = true;
                            break;
                        }
                    }
                    break;
                }
                else if (thisChar != thouSep)
                {
                    handled = true;
                }
                break;
        }

        if (!handled)
        {
            base.OnKeyPress(e);
        }
        else
        {
            e.Handled = true;
        }

    }
    protected override void OnLeave(EventArgs e)
    {
        string tmp = this.Text;
        if (tmp == "")
        {
            tmp = "0";
            numVal = NumberLostFocus(ref tmp);
            this.Text = tmp;
        }
        if (tmp.Substring(tmp.Length - 1) == "%")
        {
            tmp = tmp.Substring(0, tmp.Length - 1);
            numVal = 0.0;
            numVal = NumberLostFocus(ref tmp) / 100.0;
            double test = numVal * 100.0;
            this.Text = FormatNumber(test) + "%";
        }
        else if (perCent)
        {
            numVal = NumberLostFocus(ref tmp);
            double test = numVal * 100.0;
            this.Text = FormatNumber(test) + "%";
        }
        else
```

```
        {
            numVal = NumberLostFocus(ref tmp);
            this.Text = tmp;
        }
        base.OnLeave(e);
    }
    private string FormatNumber(double amount)
    {
        NumberFormatInfo nF = new NumberFormatInfo();
        nF.NumberDecimalSeparator = decSep;
        nF.NumberGroupSeparator = thouSep;

        string decFormat;
        if (useThouSep)
        {
            decFormat = "#,##0";
        }
        else
        {
            decFormat = "#0";
        }
        if (decPlaces > 0)
        {
            decFormat += ".";
            for (int i = 0; i < decPlaces; i++)
            {
                decFormat += "0";
            }
            if (extraDecPlaces > 0)
            {
                for (int i = 0; i < extraDecPlaces; i++)
                {
                    decFormat += "#";
                }
            }
        }
        else if (extraDecPlaces > 0)
        {
            decFormat += ".";
            for (int i = 0; i < extraDecPlaces; i++)
            {
                decFormat += "#";
            }
        }
        return (amount.ToString(decFormat, nF));
    }
    private double NumberLostFocus(ref string amountBox)
    {
        if (amountBox.Substring(0, 1) == decSep)
            amountBox = "0" + amountBox;
        NumberFormatInfo nF = new NumberFormatInfo();
        nF.NumberDecimalSeparator = decSep;
        nF.NumberGroupSeparator = thouSep;

        try
        {
            double d = 0.0;
            int l = amountBox.Length;
            if (l > 0)
            {
```

```
            char[] c = amountBox.ToCharArray();
            char endChar = c[l - 1];

            switch (endChar)
            {
                case '0':
                case '1':
                case '2':
                case '3':
                case '4':
                case '5':
                case '6':
                case '7':
                case '8':
                case '9':
                    {
                        stripNonNumerics(ref amountBox);
                        d = Double.Parse(amountBox, nF);
                        break;
                    }
                case 'm':
                case 'M':
                    {
                        if (amountBox.Length == 1)
                            d = 1000000.0;
                        else
                        {
                            string s = amountBox.Substring(0, l - 1);
                            stripNonNumerics(ref s);
                            d = Double.Parse(s, nF) * 1000000.0;
                        }
                        break;

                    }
                case 't':
                case 'T':
                    {
                        if (amountBox.Length == 1)
                            d = 1000.0;
                        else
                        {
                            string s = amountBox.Substring(0, l - 1);
                            stripNonNumerics(ref s);
                            d = Double.Parse(s, nF) * 1000.0;
                        }
                        break;
                    }
                default:
                    {
                        //remove offending char
                        string s = amountBox.Substring(0, l - 1);
                        if (s.Length > 0)
                        {
                            stripNonNumerics(ref s);
                            d = Double.Parse(s, nF);
                        }
                        else
                            d = 0.0;
                        break;
                    }
            }
```

```
            }
            amountBox = FormatNumber(d);
            return (d);
        }
        catch (Exception e)
        {
            //handle exception here;
            return 0.0;
        }
    }
    private void stripNonNumerics(ref string amountBox)
    {
        bool dSFound = false;
        char[] tmp = decSep.ToCharArray();
        char dS = tmp[0];
        string cleanNum = "";
        int l = amountBox.Length;
        if (l > 0)
        {
            char[] c = amountBox.ToCharArray();
            for (int i = 0; i < l; i++)
            {
                char b = c[i];
                switch (b)
                {
                    case '0':
                    case '1':
                    case '2':
                    case '3':
                    case '4':
                    case '5':
                    case '6':
                    case '7':
                    case '8':
                    case '9':
                        cleanNum += b;
                        break;
                    case '-':
                        if (i == 0)
                            cleanNum += b;
                        break;
                    default:
                        if ((b == dS) && (!dSFound))
                        {
                            dSFound = true;
                            cleanNum += b;
                        }
                        break;
                }
            }
        }
        amountBox = cleanNum;
    }
}
```

Además de restringir la entrada a números, esta clase tiene algunas características especiales. Expone un valor de propiedad para representar el valor doble del número, formatea el texto, opcionalmente con miles de separadores, y proporciona entradas de números grandes a mano corta: 10M se expande en licencia a 10,000,000.00 (el número de lugares decimales es una

propiedad). Por razones de brevedad, los decimales y los miles de separadores han sido codificados. En un sistema de producción, estas son también las preferencias del usuario.

Lea Controles hereditarios en línea: https://riptutorial.com/es/winforms/topic/6476/controles-hereditarios

Capítulo 5: El enlace de datos

Parámetros

Argumento	Descripción
nombre de la propiedad	El nombre de la propiedad de control a enlazar.
fuente de datos	Un objeto que representa la fuente de datos.
miembro de datos	La propiedad o lista a enlazar.
formateo habilitado	Determina si los datos mostrados deben ser formateados.
modo de actualizacion	La fuente de datos se actualiza cuando la propiedad de control se valida (predeterminada) o inmediatamente cuando la propiedad ha cambiado
valor nulo	Cuando el origen de datos tiene este valor, la propiedad enlazada se establece en DBNull.
formatString	Uno o más caracteres de especificador de formato que indican cómo se mostrará un valor
formatInfo	Una implementación de IFormatProvider para anular el comportamiento de formato predeterminado.

Observaciones

Consulte https://msdn.microsoft.com/en-us/library/ef2xyb33.aspx El enlace de datos solo funciona con propiedades, nunca con campos.

Examples

Encuadernación de controles a objetos de datos.

Cada control tiene una propiedad `DataBindings` que es una lista de objetos `System.Windows.Forms.Binding` . El método Add () tiene algunas sobrecargas que le permiten enlazar fácilmente a la propiedad de un objeto:

```
textBox.DataBindings.Add( "Text", dataObj, "MyProperty" );
```

Tenga en cuenta que el enlace básicamente significa suscribirse a cada evento de cambio. El código anterior se suscribe al cambio de evento de dataObj.MyProperty y adapta textBox.Text cuando cambia. Y viceversa, se suscribe a textBox.TextChanged y adapta dataObj.MyPropery cuando cambia.

Lea El enlace de datos en línea: https://riptutorial.com/es/winforms/topic/7362/el-enlace-de-datos

Capítulo 6: Integración de ayuda

Observaciones

Puede proporcionar ayuda para formularios y controles en aplicaciones de formularios Windows Forms de diferentes maneras. Puede mostrar una ayuda emergente, abrir un archivo CHM o una URL. Puede mostrar ayuda contextual para formularios, controles y cuadros de diálogo.

Componente HelpProvider

Puede configurar un componente `HelpProvider` para proporcionar ayuda sensible al contexto para el componente. De esta manera, cuando el usuario presione `la` tecla `F1` o el botón de Ayuda de la forma, puede automáticamente:

- Mostrar una ventana emergente de ayuda sensible al contexto para los controles
- Abra un archivo CHM según el contexto (Mostrar tabla de contenido, Mostrar una palabra clave o índice, mostrar un tema)
- Navegue a una URL usando el navegador predeterminado

Clase de ayuda

Puede usar la clase de `Help` en el código, para proporcionar este tipo de ayuda:

- Mostrar una ayuda emergente para un control
- Abra un archivo CHM según el contexto (Mostrar tabla de contenido, Mostrar una palabra clave o índice, mostrar un tema)
- Navegue a una URL usando el navegador predeterminado

Evento de Solicitud de Ayuda

Puede manejar el evento `HelpRequested` de objetos de `Control` o `Form` para realizar acciones personalizadas cuando el usuario presione `F1` o haga clic en el botón Ayuda del formulario.

Botón de ayuda de la forma

Puede configurar el `Form` para que muestre el botón Ayuda en la barra de título. De esta manera, si el usuario hace clic en el botón Ayuda, el cursor cambiará a `?` y después de hacer clic en cualquier punto, se mostrará cualquier ayuda sensible al contexto asociada con el control que usa el `HelpProvider` .

Botón de ayuda de MessgeBox y CommonDialogs

Puede proporcionar ayuda para `MessageBox` , `OpenFileDialog` , `SaveDialog` y `ColorDialog` usando el

botón de Ayuda de los componentes.

Componente ToolTip

Puede usar el componente `ToolTip` para mostrar un texto de ayuda cuando el usuario apunta a los controles. Una `ToolTip` se puede asociar con cualquier control.

Nota

Uso de `HelpProvider` y clase de `Help` Puede mostrar archivos de Ayuda compilados (.chm) o archivos HTML en el formato de Ayuda HTML. Los archivos de Ayuda compilados proporcionan una tabla de contenido, un índice, capacidad de búsqueda y enlaces de palabras clave en las páginas. Los accesos directos solo funcionan en los archivos de Ayuda compilados. Puede generar archivos de Ayuda HTML 1.x utilizando el Taller de Ayuda HTML. Para obtener más información sobre la Ayuda HTML, consulte "Taller de Ayuda HTML" y otros temas de la Ayuda HTML en la Ayuda HTML de Microsoft .

Examples

Mostrar archivo de ayuda

La `Help Class` encapsula el motor de ayuda 1.0 de HTML. Puede usar el objeto de Ayuda para mostrar los archivos de Ayuda compilados (.chm) o los archivos HTML en el formato de Ayuda HTML. Los archivos de Ayuda compilados proporcionan enlaces de tabla de contenido, índice, búsqueda y palabras clave en las páginas. Los accesos directos solo funcionan en los archivos de Ayuda compilados. Puede generar archivos de Ayuda HTML 1.x con una herramienta gratuita de Microsft llamada `HTML Help Workshop` .

Una forma fácil de mostrar un archivo de ayuda compilado en una segunda ventana:

DO#

```
Help.ShowHelp(this, helpProviderMain.HelpNamespace);
```

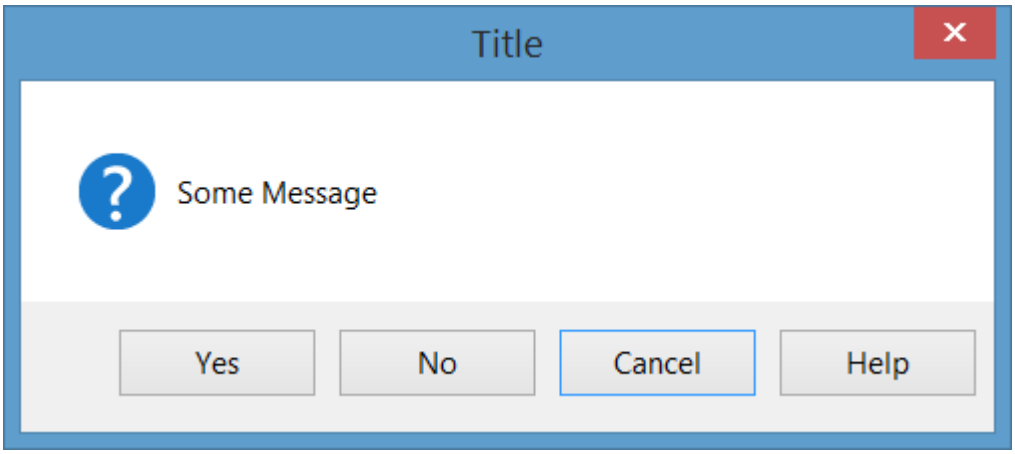
VB.NET

```
Help.ShowHelp(Me, hlpProviderMain.HelpNamespace)
```

Mostrar ayuda para MessageBox

Puede proporcionar ayuda para el cuadro de mensaje de diferentes maneras. Puede configurar un `MessageBox` para mostrar un botón de `Help` o no. También puede configurar `MessageBox` de manera que cuando el usuario solicite ayuda haciendo clic en el botón Ayuda o presionando `F1` , muestre un archivo CHM o navegue a una URL o realice una acción personalizada. Aquí hay algunos ejemplos en este tema.

En todos los ejemplos a continuación, el `MessageBox` sería así:



Mostrar un archivo CHM y navegar a una palabra clave (índice)

```
MessageBox.Show("Some Message", "Title", MessageBoxButtons.YesNoCancel,
    MessageBoxIcon.Question, MessageBoxDefaultButton.Button3, 0,
    "help.chm", HelpNavigator.KeywordIndex, "SomeKeyword");
```

Mostrar un archivo CHM y navegar a un tema

```
MessageBox.Show("Some Message", "Title", MessageBoxButtons.YesNoCancel,
    MessageBoxIcon.Question, MessageBoxDefaultButton.Button3, 0,
    "help.chm", HelpNavigator.Topic, "/SomePath/SomePage.html");
```

Muestra un archivo CHM y navega por la primera página de ayuda en la tabla de contenido

```
MessageBox.Show("Some Message", "Title", MessageBoxButtons.YesNoCancel,
    MessageBoxIcon.Question, MessageBoxDefaultButton.Button3, 0,
    "help.chm");
```

Abre el navegador predeterminado y navega a una URL

```
MessageBox.Show("Some Message", "Title", MessageBoxButtons.YesNoCancel,
    MessageBoxIcon.Question, MessageBoxDefaultButton.Button3, 0,
    "http://example.com");
```

Realizar una acción personalizada al presionar el botón de Ayuda o la tecla F1

En este caso, debe manejar el evento `HelpRequested` del padre de `MessageBox` y realizar una operación personalizada:

```
private void Form1_HelpRequested(object sender, HelpEventArgs hlpevent)
{
    // Perform custom action, for example show a custom help form
    var f = new Form();
    f.ShowDialog();
}
```

Luego puedes mostrar el `MessageBox` con el botón de Ayuda:

```
MessageBox.Show("Some Message", "Title", MessageBoxButtons.YesNoCancel,
    MessageBoxIcon.Question, MessageBoxDefaultButton.Button3, 0, true);
```

O mostrarlo sin el botón de ayuda:

```
MessageBox.Show("Some Message", "Title", MessageBoxButtons.YesNoCancel,
    MessageBoxIcon.Question, MessageBoxDefaultButton.Button3, 0, false);
```

Mostrar ayuda para CommonDialogs

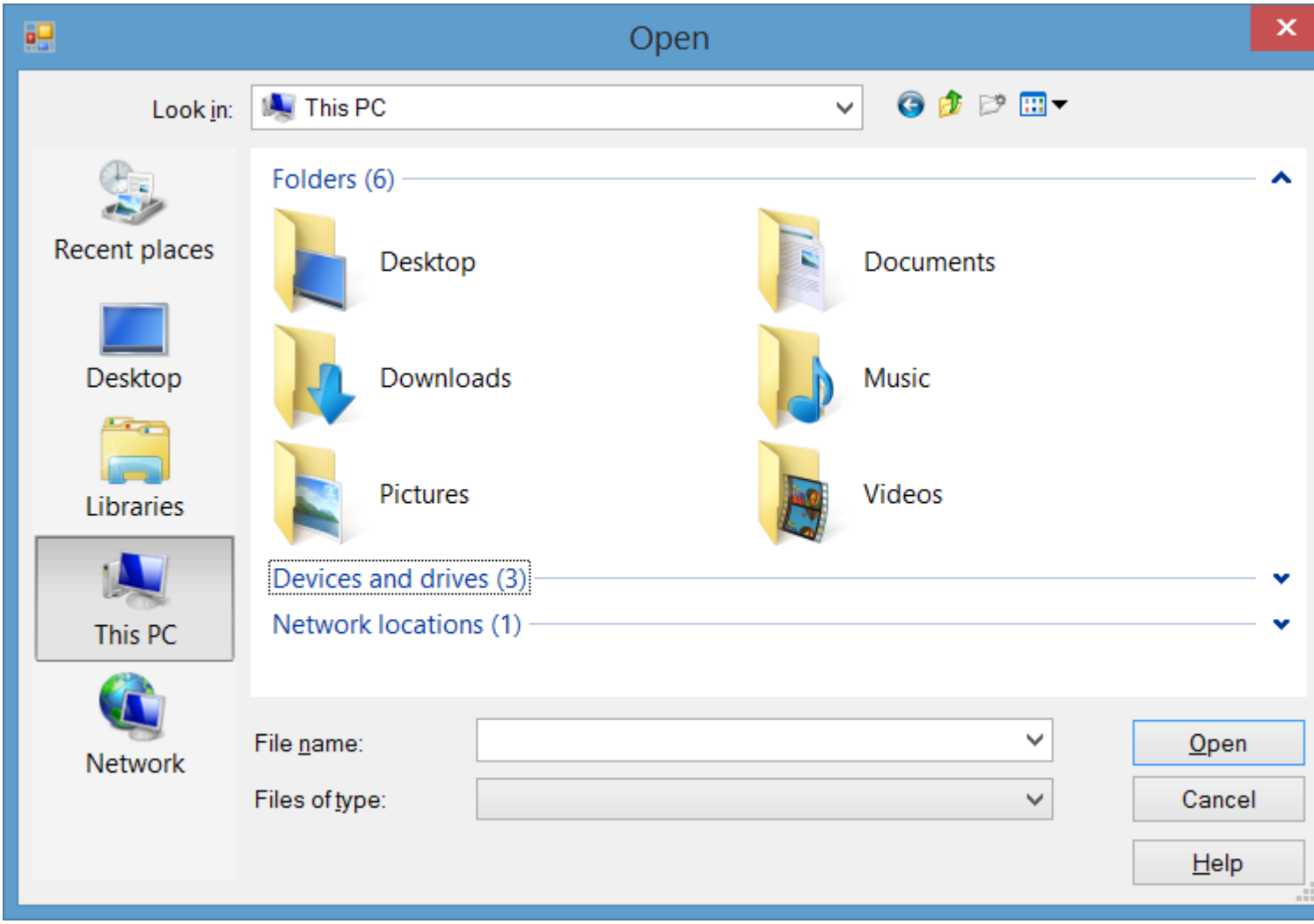
Puede proporcionar ayuda para `OpenFileDialog` , `SaveFileDialog` y `ColorDialog` . Para hacerlo, establezca la propiedad `ShowHelp` del diálogo en `true` y maneje el evento `HelpRequest` para el diálogo:

```
void openFileDialog1_HelpRequest(object sender, EventArgs e)
{
    //Perform custom action
    Help.ShowHelp(this, "Http://example.com");
}
```

Nota

- El evento se generará solo si establece `ShowHelp` en `true` .
- El evento se levantará solo haciendo clic en el botón `Help` y no se levantará usando la tecla F1.

En la imagen de abajo puedes ver un `OpenFileDialog` con un botón de Ayuda:



Manejando el evento HelpRequested de Controls and Form

Cuando un usuario presiona `F1` en un control o hace clic en el botón Ayuda de la forma (`?`) Y luego hace clic en un control, se `HelpRequested` evento `HelpRequested` .

Puede manejar este evento para proporcionar una acción personalizada cuando el usuario solicite ayuda para los controles o el formulario.

El `HelpRequested` soporta el mecanismo de burbuja. Se dispara para su control activo y si no controla el evento y no establece la propiedad `Handled` de su evento arg en `true` , entonces se eleva hasta la jerarquía de control principal hasta la forma.

Por ejemplo, si maneja el evento `HelpRequested` del formulario como se muestra a continuación, al presionar `F1` aparecerá un cuadro de mensaje que mostrará el nombre del control activo, pero para `textBox1` mostrará un mensaje diferente:

```
private void Form1_HelpRequested(object sender, HelpEventArgs hlpevent)
{
    var c = this.ActiveControl;
    if(c!=null)
        MessageBox.Show(c.Name);
}
private void textBox1_HelpRequested(object sender, HelpEventArgs hlpevent)
```

```
{
    hlpevent.Handled = true;
    MessageBox.Show("Help request handled and will not bubble up");
}
```

Puede realizar cualquier otra acción personalizada, como navegar a una URL o mostrar un archivo CHM utilizando la clase de `Help` .

Mostrar ayuda usando la clase de ayuda

Puede usar la clase de `Help` en el código, para proporcionar este tipo de ayuda:

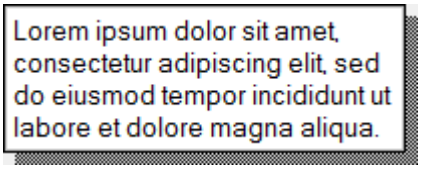
- Mostrar una ayuda emergente para un control
- Abra un archivo CHM según el contexto (Mostrar tabla de contenido, Mostrar una palabra clave o índice, mostrar un tema)
- Navegue a una URL usando el navegador predeterminado

Mostrar ventana emergente de ayuda

Puede usar `Help.ShowPopup` para mostrar una ventana emergente de ayuda:

```
private void control_MouseClick(object sender, MouseEventArgs e)
{
    var c = (Control)sender;
    var help = "Lorem ipsum dolor sit amet, consectetur adipiscing elit, " +
               "sed do eiusmod tempor incididunt ut labore et dolore magna aliqua."
    if (c != null)
        Help.ShowPopup(c, "Lorem ipsum dolor sit amet.", c.PointToScreen(e.Location));
}
```

Mostrará tal ayuda emergente en la ubicación del puntero del mouse:



Mostrar archivo de ayuda CHM

Puede usar diferentes sobrecargas del método `Help.ShowHelp` para mostrar un archivo CHM y navegar a una palabra clave, un tema, índice o tabla de contenido:

Mostrar tabla de ayuda de contenido

```
Help.ShowHelp(this, "Help.chm");
```

Mostrar ayuda para palabra clave específica (índice)

```
Help.ShowHelp(this, "Help.chm", HelpNavigator.Index, "SomeKeyword");
```

Mostrar ayuda para un tema específico

```
Help.ShowHelp(this, "Help.chm", HelpNavigator.Topic, "/SomePath/SomePage.html");
```

Mostrar url

Puedes mostrar cualquier URL en el navegador predeterminado usando el método `ShowHelp` :

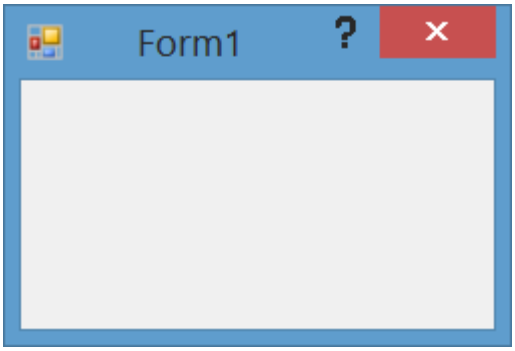
```
Help.ShowHelp(this, "Http://example.com");
```

Mostrar botón de ayuda en la barra de título del formulario

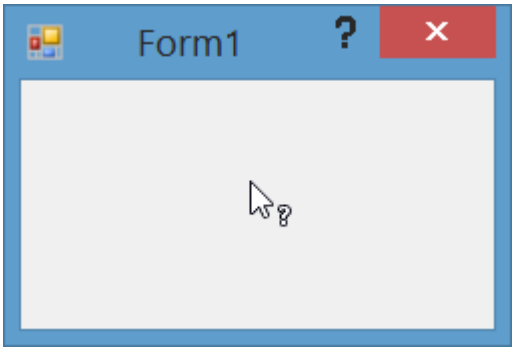
Puede mostrar un botón de ayuda en la barra de título de un `Form` . Para hacerlo, debes:

1. Establezca la propiedad `HelpButton` de form en `true` .
2. Establezca `MinimizeBox` y `MaximizeBox` en `false` .

Luego aparecerá un botón de ayuda en la barra de título de la `Form` :



Además, al hacer clic en el botón Ayuda, el cursor cambiará a `?` cursor:



Luego, si hace clic en un `Control` o `Form` , se `HelpRequested` evento `HelpRequested` y también si ha configurado un `HelpProvider` , la ayuda para el control se mostrará usando `HelpProvider` .

Crea un botón de Ayuda personalizado que actúa como el Formulario de Ayuda estándar.

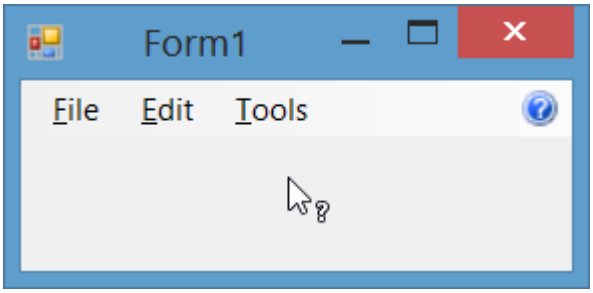
Si tiene un `Form` con `MinimizeBox` y `MaximizeBox` establecido en `true` , entonces no puede mostrar el botón Ayuda en la barra de título de la `Form` y perderá la función de hacer clic en el botón de ayuda para convertirlo en el cursor de ayuda para poder hacer clic en los controles para mostrar ayuda.

Puede hacer que un elemento del menú en `MenuStrip` actúe como el botón de Ayuda estándar. Para hacerlo, agregue un `MenuStrip` al formulario y agregue un `ToolStripMenuItem` al mismo, luego maneje el evento `Click` del elemento:

```
private const int WM_SYSCOMMAND = 0x0112;
private const int SC_CONTEXTHELP = 0xF180;
[System.Runtime.InteropServices.DllImport("user32.dll")]
static extern IntPtr SendMessage(IntPtr hWnd, int Msg, int wParam, int lParam);
private void helpToolStripMenuItem_Click(object sender, EventArgs e)
{
    SendMessage(this.Handle, WM_SYSCOMMAND, SC_CONTEXTHELP, 0);
}
```

Nota: Si desea hacerlo usando un `Button` , también necesita configurar `button1.Capture = false;` antes de enviar el mensaje. Pero no es necesario para un `ToolStripMenuItem` .

Luego, al hacer clic en el menú de ayuda, el cursor cambiará a `?` cursor y actuará como cuando haces clic en el botón de Ayuda estándar:



Manejo del evento HelpButtonClicked de Form

Puede detectar cuándo un usuario hizo `HelpButton` en un `HelpButton` en la barra de título de la forma manejando `HelpButtonClicked` . Puede dejar que el evento continúe o cancelarlo configurando la propiedad `Cancel` de sus argumentos de evento en `true` .

```
private void Form1_HelpButtonClicked(object sender, CancelEventArgs e)
{
    e.Cancel = true;
    //Perform some custom action
    MessageBox.Show("Some Custom Help");
}
```

Lea Integración de ayuda en línea: https://riptutorial.com/es/winforms/topic/3285/integracion-de-ayuda

Capítulo 7: Mostrando un formulario

Introducción

Este tema explica cómo funciona el motor de WinForms para mostrar formularios y cómo controla sus tiempos de vida.

Examples

Muestra una forma modelo o modal.

Después de definir la estructura de su formulario con el diseñador de WinForms, puede mostrar sus formularios en código con dos métodos diferentes.

- Método - Una forma Modera

```
Form1 aForm1Instance = new Form1();
aForm1Instance.Show();
```

- Método - Un diálogo modal

```
Form2 aForm2Instance = new Form2();
aForm2Instance.ShowDialog();
```

Los dos métodos tienen una distinción muy importante. El primer método (el modelo) muestra su formulario y luego regresa inmediatamente sin esperar el cierre del formulario recién abierto. Entonces tu código continúa con lo que sigue a la llamada Show. En su lugar, el segundo método (el modal) abre el formulario y bloquea cualquier actividad en toda la aplicación hasta que cierre el formulario con el botón de cerrar o con algunos botones configurados adecuadamente para cerrar el formulario.

Cerrar una forma de modelo.

Normalmente se emplea un formulario sin modelo cuando necesita mostrar algo de forma permanente a lo largo de la pantalla principal de la aplicación (piense en una leyenda o una vista en un flujo de datos provenientes de un dispositivo o una ventana secundaria de MDI).
Pero una forma sin modelo plantea un desafío único cuando se quiere cerrar. ¿Cómo recuperar la instancia y llamar al método Close en esa instancia?

Puede mantener una variable global que haga referencia a la instancia que desea cerrar.

```
theGlobalInstance.Close();
theGlobalInstance.Dispose();
theGlobalInstance = null;
```

Pero también podemos optar por utilizar la colección Application.OpenForms donde el motor de

formulario almacena todas las instancias de formulario creadas y aún abiertas.

Puede recuperar esa instancia en particular de esta colección y llamar al método Close

```
Form2 toClose = Application.OpenForms.OfType<Form2>().FirstOrDefault();
if(toClose != null)
{
    toClose.Close();
    toClose.Dispose();
}
```

Cierre de un formulario modal.

Cuando se muestra un formulario utilizando el método `ShowDialog` , es necesario configurar la propiedad `DialogResult` del formulario para que se cierre. Esta propiedad se puede establecer utilizando la enumeración que también se llama DialogResult .

Para cerrar un formulario, solo necesita establecer la propiedad `DialogResult` del formulario (a cualquier valor de `DialogResult.None`) en algún controlador de eventos. Cuando su código salga del controlador de eventos, el motor de WinForm ocultará el formulario y el código que sigue a la llamada inicial `ShowDialog` método `ShowDialog` continuará su ejecución.

```
private cmdClose_Click(object sender, EventArgs e)
{
    this.DialogResult = DialogResult.Cancel;
}
```

El código de llamada puede capturar el valor de retorno de ShowDialog para determinar en qué botón hizo clic el usuario en el formulario. Cuando se muestra con `ShowDialog()` , el formulario no se elimina automáticamente (ya que simplemente se ocultó y no se cerró), por lo que es importante usar un bloque de `using` para garantizar que se elimine el formulario.

A continuación se muestra un ejemplo de `OpenFileDialog` verificar el resultado de usar `OpenFileDialog` , verificar el resultado y acceder a una propiedad desde el cuadro de diálogo antes de desecharlo.

```
using (var form = new OpenFileDialog())
{
    DialogResult result = form.ShowDialog();
    if (result == DialogResult.OK)
    {
        MessageBox.Show("Selected file is: " + form.FileName);
    }
}
```

También puede establecer la propiedad `DialogResult` en un botón. Al hacer clic en ese botón se establecerá la propiedad `DialogResult` en el formulario al valor asociado con el botón. Esto le permite cerrar el formulario sin agregar un controlador de eventos para establecer `DialogResult` en el código.

Por ejemplo, si agrega un botón Aceptar a su formulario y establece su propiedad en

`DialogResult.OK` , el formulario se cierra automáticamente cuando presiona ese botón y el código de llamada recibe un `DialogResult.OK` a cambio de la llamada del método `ShowDialog()` .

Lea Mostrando un formulario en línea: https://riptutorial.com/es/winforms/topic/8768/mostrando-un-formulario

Creditos

S. No	Capítulos	Contributors
1	Primeros pasos con winforms	4444, Bjørn-Roger Kringsjå, Chris Shao, Cody Gray, Community, Reza Aghaei
2	Caja de texto	gplumb, Jones Joseph, Stefano d'Antonio
3	Controles basicos	Aimnox, Berkay, help-info.de, Jeff Bridgman
4	Controles hereditarios	Balagurunathan Marimuthu
5	El enlace de datos	Kai Thoma
6	Integración de ayuda	help-info.de, Reza Aghaei
7	Mostrando un formulario	Cody Gray, Jeff Bridgman, Steve