



EBook Gratuito

APPENDIMENTO

winforms

Free unaffiliated eBook created from
Stack Overflow contributors.

#winforms

Sommario

Di.....	1
Capitolo 1: Iniziare con Winforms.....	2
Osservazioni.....	2
Guarda anche:.....	2
Examples.....	2
Creazione di un'applicazione Simple WinForms utilizzando Visual Studio.....	2
Crea un progetto Windows Form.....	3
Aggiungi controlli al modulo.....	3
Scrivi codice.....	4
Esegui e prova.....	5
Creazione di un'applicazione C # WinForms semplice utilizzando un editor di testo.....	5
Creazione di un'applicazione WinForms VB.NET semplice mediante un editor di testo.....	6
Capitolo 2: Aiuta l'integrazione.....	9
Osservazioni.....	9
HelpProvider Component.....	9
Aiuta classe.....	9
Aiuto Evento Richiesto.....	9
Help Button of Form.....	9
Pulsante Aiuto di MessgeBox e CommonDialogs.....	9
Componente descrizione.....	10
Examples.....	10
Mostra il file di aiuto.....	10
Mostra la Guida per MessageBox.....	10
Mostra un file CHM e vai a una parola chiave (indice).....	11
Mostra un file CHM e vai a un argomento.....	11
Mostra un file CHM e naviga nella prima pagina di aiuto nel sommario.....	11
Apri il browser predefinito e naviga verso un URL.....	11
Esegui un'azione personalizzata quando premi il pulsante Guida o il tasto F1.....	11
Mostra Guida per CommonDialogs.....	12
Gestione della guida Evento richiesto di controlli e modulo.....	13

Mostra Guida utilizzando la classe della Guida.....	14
Mostra la finestra a comparsa Aiuto.....	14
Mostra il file della Guida CHM.....	14
Mostra il sommario della guida.....	14
Mostra guida per parole chiave specifiche (indice).....	14
Mostra guida per argomento specifico.....	15
Mostra URL.....	15
Mostra il pulsante Guida sulla barra del titolo del modulo.....	15
Crea un pulsante di Guida personalizzato che funziona come HelpButton Form standard.....	15
Gestione di HelpButtonErrore evento di modulo.....	16
Capitolo 3: Associazione dati.....	17
Parametri.....	17
Osservazioni.....	17
Examples.....	17
Controllo dei collegamenti agli oggetti dati.....	17
Capitolo 4: Casella di testo.....	19
Examples.....	19
Completamento automatico da una raccolta di stringhe.....	19
Consenti solo cifre nel testo.....	19
Come scorrere fino alla fine.....	19
Aggiunta di un segnaposto alla casella di testo.....	20
Capitolo 5: Controlli di base.....	21
Examples.....	21
Pulsante.....	21
Casella di testo.....	21
Casella combinata.....	22
CheckBox.....	24
ListBox.....	24
NumericUpDown.....	28
Eventi utili.....	30
Capitolo 6: Controlli ereditari.....	32
Osservazioni.....	32

Examples.....	32
Impostazioni a livello di applicazione.....	32
NumberBox.....	35
Capitolo 7: Mostrare un modulo.....	43
introduzione.....	43
Examples.....	43
Mostra una forma modale o modale.....	43
Chiusura di un modulo non modale.....	43
Chiusura di una forma modale.....	44
Titoli di coda.....	46

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.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

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

Capitolo 1: Iniziare con Winforms

Osservazioni

Windows Form (in breve "WinForms") è una libreria di classi GUI inclusa in .NET Framework. Si tratta di un sofisticato wrapper orientato agli oggetti attorno [all'API Win32](#), che consente lo sviluppo di applicazioni desktop e mobili Windows destinate a [.NET Framework](#).

WinForms è principalmente [basato sugli eventi](#). Un'applicazione è costituita da più *moduli* (visualizzati come finestre sullo schermo), che contengono *controlli* (etichette, pulsanti, caselle di testo, elenchi, ecc.) Con cui l'utente interagisce direttamente. In risposta all'interazione dell'utente, questi controlli generano eventi che possono essere gestiti dal programma per eseguire attività.

Come in Windows, tutto in WinForms è un controllo, che è di per sé un tipo di finestra. La classe Control base fornisce funzionalità di base, incluse le proprietà per l'impostazione di testo, posizione, dimensione e colore, nonché un insieme comune di eventi che possono essere gestiti. Tutti i controlli derivano dalla classe Control, aggiungendo funzionalità aggiuntive. Alcuni controlli possono ospitare altri controlli, sia per la riutilizzabilità (`Form`, `UserControl`) che per il layout (`TableLayoutPanel`, `FlowLayoutPanel`).

WinForms è stato supportato dalla versione originale di .NET Framework (v1.0) ed è ancora disponibile nelle versioni moderne (v4.5). Tuttavia, non è più in fase di sviluppo attivo e non vengono aggiunte nuove funzionalità. [Secondo 9 sviluppatori Microsoft alla conferenza Build 2014](#) :

Windows Form continua a essere supportato, ma in modalità di manutenzione. Risolveranno i bug man mano che vengono scoperti, ma la nuova funzionalità è fuori dal tavolo.

La [libreria Mono multi](#) -piattaforma e open-source fornisce un'implementazione di base di Windows Form, supportando tutte le funzionalità implementate da Microsoft a partire da .NET 2.0. Tuttavia, WinForms non è sviluppato attivamente su Mono e un'implementazione completa è considerata impossibile, dato il legame indissolubile tra il framework e l'API Windows nativa (che non è disponibile in altre piattaforme).

Guarda anche:

- Documentazione [Windows Form](#) su MSDN

Examples

Creazione di un'applicazione Simple WinForms utilizzando Visual Studio

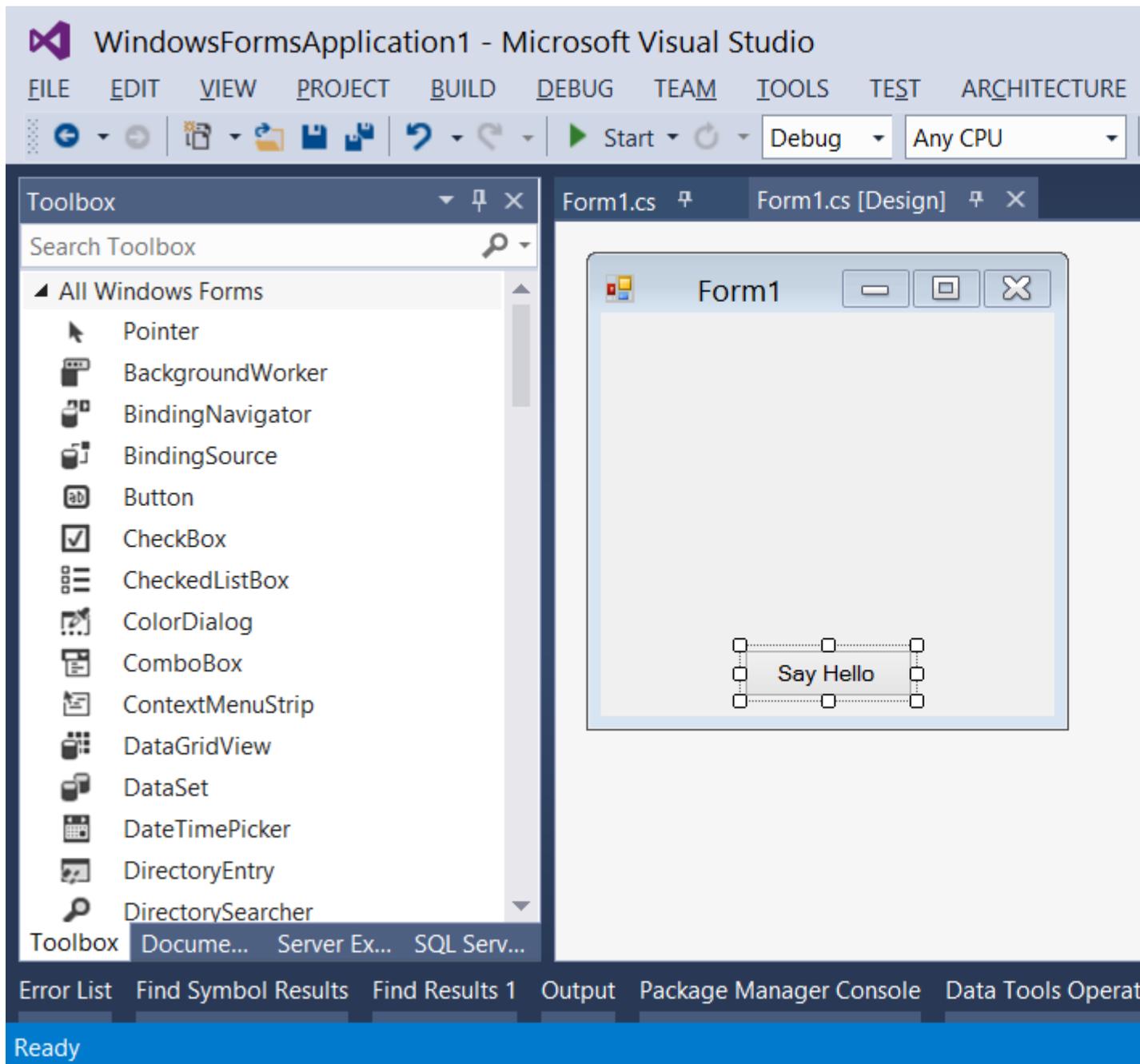
Questo esempio ti mostrerà come creare un progetto di applicazione Windows Form in Visual Studio.

Crea un progetto Windows Form

1. Avvia Visual Studio.
2. Dal menu **File** , scegliere **Nuovo** e quindi selezionare **Project** . Viene visualizzata la finestra di dialogo **Nuovo progetto** .
3. Nel riquadro **Modelli installati** , selezionare "Visual C #" o "Visual Basic".
4. Sopra il riquadro centrale, è possibile selezionare il framework di destinazione dall'elenco a discesa.
5. Nel riquadro centrale, selezionare il modello di **applicazione Windows Form** .
6. Nella casella di testo **Nome** , digitare un nome per il progetto.
7. Nella casella di testo **Posizione** , selezionare una cartella per salvare il progetto.
8. Clicca **OK** .
9. La Progettazione Windows Form si apre e visualizza **Form1** del progetto.

Aggiungi controlli al modulo

1. Dalla palette degli **strumenti** , trascinare un controllo **Button** sul modulo.
2. Fare clic sul pulsante per selezionarlo. Nella finestra Proprietà, impostare la proprietà `Text` su **Say Hello** .



Scrivi codice

1. Fare doppio clic sul pulsante per aggiungere un gestore di eventi per l'evento `Click`. L'editor di codice si aprirà con il punto di inserimento inserito all'interno della funzione del gestore di eventi.
2. Digita il seguente codice:

C

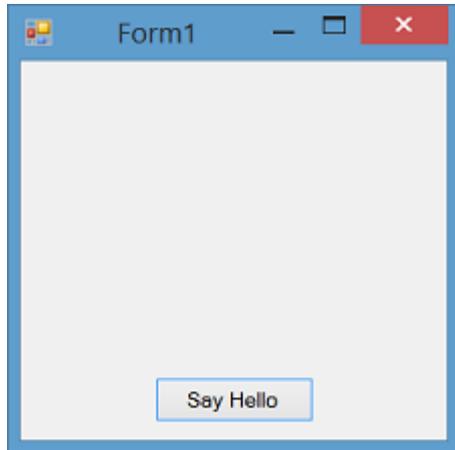
```
MessageBox.Show("Hello, World!");
```

VB.NET

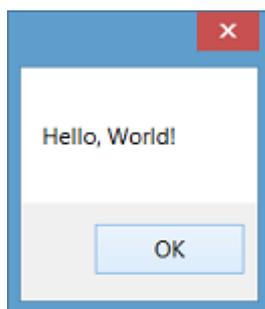
```
MessageBox.Show("Hello, World!");
```

Esegui e prova

1. Premere **F5** per eseguire l'applicazione.



2. Quando la tua applicazione è in esecuzione, fai clic sul pulsante per vedere "Hello, World!" Messaggio.



3. Chiudere il modulo per tornare a Visual Studio.

Creazione di un'applicazione C # WinForms semplice utilizzando un editor di testo

1. Apri un editor di testo (come Blocco note) e digita il codice seguente:

```
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. Salva il file in un percorso a cui hai accesso in lettura / scrittura. È convenzionale nominare il file dopo la classe che contiene, ad esempio `X:\MainForm.cs`.
3. Esegui il compilatore C # dalla riga di comando, passando il percorso del file di codice come argomento:

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

Nota: per utilizzare una versione del compilatore C # per altre versioni di .NET Framework, dare un'occhiata al percorso, `%WINDIR%\Microsoft.NET` e modificare l'esempio sopra di conseguenza. Per ulteriori informazioni sulla compilazione di applicazioni C #, vedere [Compilare ed eseguire il primo programma C #](#).

4. Una volta completata la compilazione, verrà creata un'applicazione denominata `MainForm.exe` nella stessa directory del file di codice. È possibile eseguire questa applicazione dalla riga di comando o facendo doppio clic su di essa in Esplora risorse.

Creazione di un'applicazione WinForms VB.NET semplice mediante un editor di testo

1. Apri un editor di testo (come Blocco note) e digita il codice seguente:

```
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. Salva il file in un percorso a cui hai accesso in lettura / scrittura. È convenzionale nominare il file dopo la classe che contiene, ad esempio `X:\MainForm.vb`.
3. Esegui il compilatore VB.NET dalla riga di comando, passando il percorso del file di codice come argomento:

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

Nota: per utilizzare una versione del compilatore VB.NET per altre versioni di .NET Framework, dare un'occhiata al percorso `%WINDIR%\Microsoft.NET` e modificare l'esempio sopra di conseguenza. Per ulteriori informazioni sulla compilazione di applicazioni VB.NET,

vedere [Hello World](#) .

4. Una volta completata la compilazione, verrà creata un'applicazione denominata `MainForm.exe` nella stessa directory del file di codice. È possibile eseguire questa applicazione dalla riga di comando o facendo doppio clic su di essa in Esplora risorse.

Leggi Iniziare con Winforms online: <https://riptutorial.com/it/winforms/topic/1018/iniziare-con-winforms>

Capitolo 2: Aiuta l'integrazione

Osservazioni

È possibile fornire assistenza per moduli e controlli in un'applicazione Windows Form in modi diversi. Puoi mostrare una guida pop-up, aprire un file CHM o un URL. È possibile visualizzare la guida sensibile al contesto per moduli, controlli e finestre di dialogo.

HelpProvider Component

È possibile impostare un componente `HelpProvider` per fornire una guida sensibile al contesto per il componente. In questo modo quando l'utente preme il tasto `F1` o il pulsante Guida del modulo, è possibile:

- Mostra una finestra di aiuto sensibile al contesto per i controlli
- Aprire un file CHM in base al contesto (Mostra tabella dei contenuti, Mostra una parola chiave o un indice, mostra un argomento)
- Passare a un URL utilizzando il browser predefinito

Aiuta classe

È possibile utilizzare la classe `Help` nel codice per fornire questo tipo di aiuto:

- Mostra un pop-up di aiuto per un controllo
- Aprire un file CHM in base al contesto (Mostra tabella dei contenuti, Mostra una parola chiave o un indice, mostra un argomento)
- Passare a un URL utilizzando il browser predefinito

Aiuto Evento Richiesto

È possibile gestire `HelpRequested` evento `HelpRequested` degli oggetti `Control` o `Form` per eseguire azioni personalizzate quando l'utente preme `F1` o fa clic sul pulsante Guida del modulo.

Help Button of Form

È possibile configurare il `Form` per mostrare il pulsante Guida sulla barra del titolo. In questo modo, se l'utente fa clic sul pulsante Guida, il cursore cambierà in a ? cursore e dopo aver fatto clic su un punto qualsiasi, verrà visualizzata qualsiasi guida sensibile al contesto associata al controllo utilizzando `HelpProvider`.

Pulsante Aiuto di MessageBox e CommonDialogs

È possibile fornire aiuto per `MessageBox`, `OpenFileDialog`, `SaveDialog` e `ColorDialog` utilizzando il

pulsante Guida dei componenti.

Componente descrizione

È possibile utilizzare il componente `ToolTip` per visualizzare del testo di guida quando l'utente punta ai controlli. Una `ToolTip` può essere associata a qualsiasi controllo.

Nota

Uso della classe `HelpProvider` e della `Help` È possibile visualizzare file della Guida compilati (.chm) o file HTML nel formato della Guida HTML. I file della Guida compilati forniscono un sommario, un indice, funzionalità di ricerca e collegamenti di parole chiave nelle pagine. Le scorciatoie funzionano solo nei file della Guida compilati. È possibile generare file 1.x della Guida HTML utilizzando l'HTML Help Workshop. Per ulteriori informazioni sulla Guida HTML, consultare "Workshop della Guida HTML" e altri argomenti della Guida HTML nella Guida HTML di [Microsoft](#).

Examples

Mostra il file di aiuto

La `Help Class` incapsula il motore HTML della Guida 1.0. È possibile utilizzare l'oggetto `Help` per visualizzare i file della Guida compilati (.chm) o i file HTML nel formato della Guida HTML. I file della Guida compilati forniscono sommari, indici, ricerche e collegamenti di parole chiave nelle pagine. Le scorciatoie funzionano solo nei file della Guida compilati. È possibile generare file 1.x della Guida HTML con uno strumento gratuito di Microsoft chiamato `HTML Help Workshop`.

Un modo semplice per mostrare un file della guida compilato in una seconda finestra:

C

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

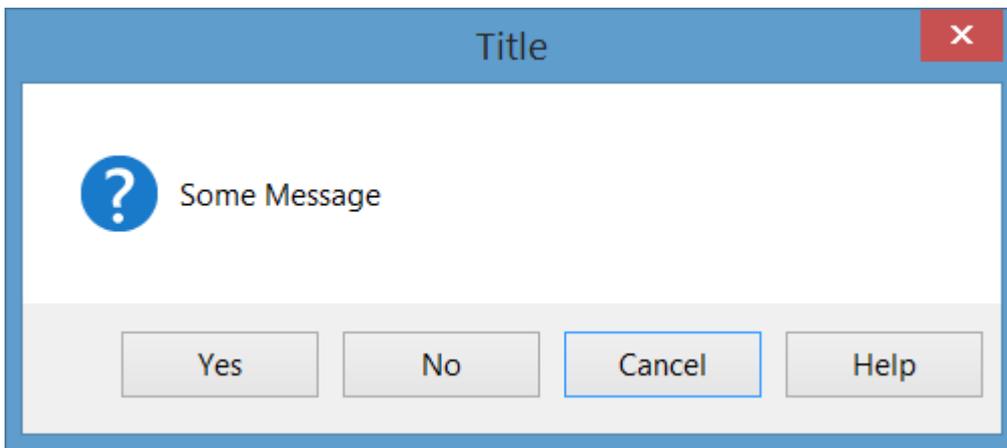
VB.NET

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

Mostra la Guida per MessageBox

È possibile fornire assistenza per la finestra di messaggio in diversi modi. È possibile configurare un `MessageBox` per mostrare o meno un pulsante `Help`. Inoltre puoi configurare `MessageBox` in modo tale che quando l'utente richiede aiuto cliccando sul pulsante Aiuto o premendo `F1`, mostra un file CHM o naviga verso un URL o esegue un'azione personalizzata. Ecco alcuni esempi in questo argomento.

In tutti gli esempi qui sotto, il `MessageBox` sarebbe così:



Mostra un file CHM e vai a una parola chiave (indice)

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

Mostra un file CHM e vai a un argomento

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

Mostra un file CHM e naviga nella prima pagina di aiuto nel sommario

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

Apri il browser predefinito e naviga verso un URL

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

Esegui un'azione personalizzata quando premi il pulsante Guida o il tasto F1

In questo caso dovresti gestire `HelpRequested` evento `HelpRequested` di parent di `MessageBox` ed eseguire operazioni personalizzate:

```
private void Form1_HelpRequested(object sender, HelpEventArgs hlpevent)
```

```
{
    // Perform custom action, for example show a custom help form
    var f = new Form();
    f.ShowDialog();
}
```

Quindi puoi mostrare il `MessageBox` con il pulsante Aiuto:

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

O mostralo senza pulsante Guida:

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

Mostra Guida per CommonDialogs

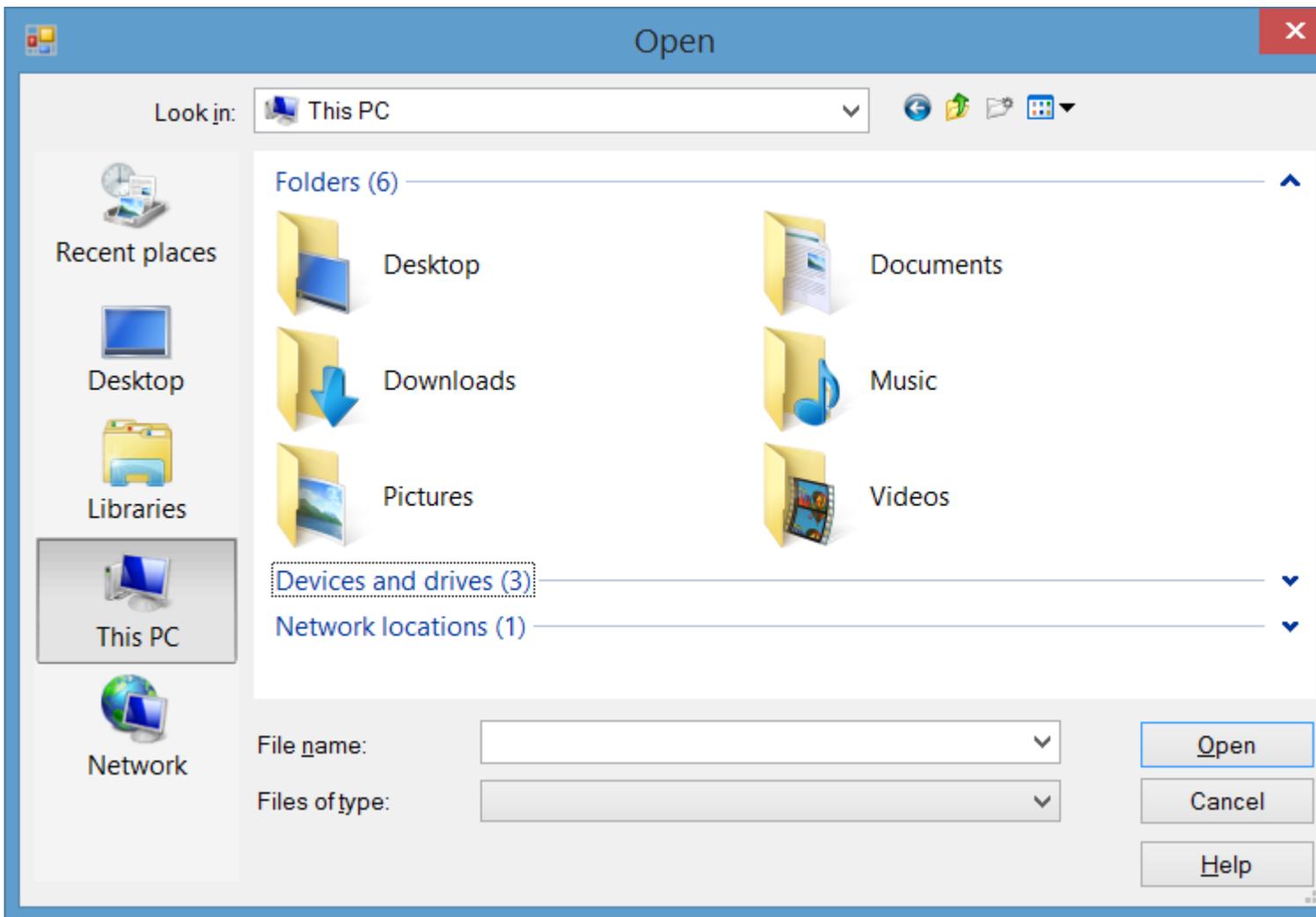
È possibile fornire aiuto per `OpenFileDialog`, `SaveFileDialog` e `ColorDialog`. Per fare ciò, imposta la proprietà `ShowHelp` della finestra di dialogo su `true` e `HelpRequest` evento `HelpRequest` per la finestra di dialogo:

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

Nota

- L'evento verrà generato solo se si imposta `ShowHelp` su `true`.
- L'evento verrà generato solo facendo clic sul pulsante `Help` e non aumenterà utilizzando il tasto F1.

Nell'immagine sottostante puoi vedere un `OpenFileDialog` con un pulsante Guida:



Gestione della guida Evento richiesto di controlli e modulo

Quando un utente preme **F1** su un controllo o fa clic sul pulsante Guida del modulo (?) E quindi fa clic su un controllo, verrà `HelpRequested` evento `HelpRequested` .

È possibile gestire questo evento per fornire un'azione personalizzata quando l'utente richiede assistenza per controlli o moduli.

`HelpRequested` supporta il meccanismo di bolla. Si attiva per il controllo attivo e se non si gestisce l'evento e non si imposta la proprietà `Handled` del relativo evento `arg` su `true` , viene visualizzata la gerarchia di controllo padre fino alla sua forma.

Ad esempio, se `HelpRequested` evento `HelpRequested` del modulo come di seguito, quando premi **F1** apparirà una finestra di messaggio e mostrerà il nome del controllo attivo, ma per `textBox1` mostrerà un messaggio diverso:

```
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");
}
```

È possibile eseguire qualsiasi altra azione personalizzata come utilizzare la navigazione verso un URL o mostrare un file CHM usando la classe [Help](#) .

Mostra Guida utilizzando la classe della Guida

È possibile utilizzare la classe [Help](#) nel codice per fornire questo tipo di aiuto:

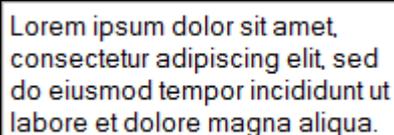
- Mostra un pop-up di aiuto per un controllo
- Aprire un file CHM in base al contesto (Mostra tabella dei contenuti, Mostra una parola chiave o un indice, mostra un argomento)
- Passare a un URL utilizzando il browser predefinito

Mostra la finestra a comparsa Aiuto

Puoi usare [Help.ShowPopup](#) per visualizzare una finestra pop-up di aiuto:

```
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));
}
```

Mostrerà tale pop-up di aiuto nella posizione del puntatore del mouse:



>Lorem ipsum dolor sit amet,
consectetur adipiscing elit, sed
do eiusmod tempor incididunt ut
labore et dolore magna aliqua.

Mostra il file della Guida CHM

Puoi utilizzare diversi overload del metodo [Help.ShowHelp](#) , per mostrare un file CHM e navigare verso una parola chiave, un argomento, un indice o una tabella di contenuti:

Mostra il sommario della guida

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

Mostra guida per parole chiave specifiche (indice)

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

Mostra guida per argomento specifico

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

Mostra URL

Puoi mostrare qualsiasi URL nel browser predefinito utilizzando il metodo `ShowHelp` :

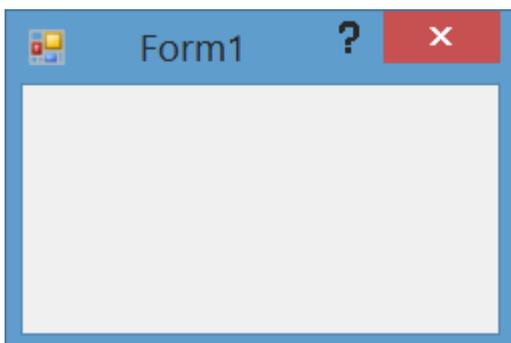
```
Help.ShowHelp(this, "Http://example.com");
```

Mostra il pulsante Guida sulla barra del titolo del modulo

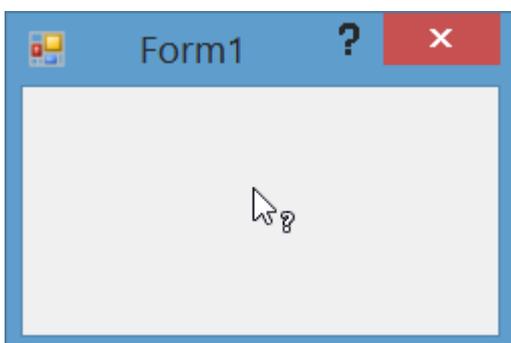
È possibile mostrare un pulsante Guida alla barra del titolo di un `Form` . Per fare ciò, dovresti:

1. Impostare la proprietà `HelpButton` del modulo su `true` .
2. Imposta `MinimizeBox` e `MaximizeBox` su `false` .

Quindi un pulsante di aiuto apparirà sulla barra del titolo di `Form` :



Inoltre, quando fai clic sul pulsante Guida, il cursore verrà modificato in un `?` cursore:



Quindi, se si fa clic su un `Control` o `Form` , verrà `HelpRequested` evento `HelpRequested` e anche se è stato impostato un `HelpProvider` , la guida per il controllo verrà visualizzata utilizzando `HelpProvider` .

Crea un pulsante di Guida personalizzato che funziona come `HelpButton`

Form standard

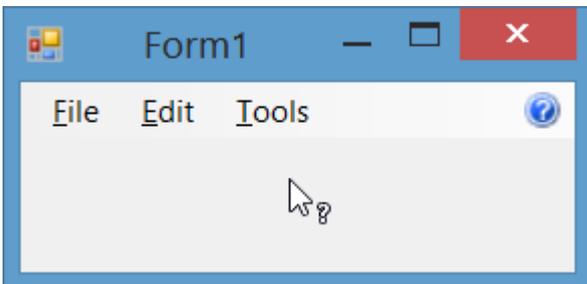
Se hai un `Form` con `MinimizeBox` e `MaximizeBox` impostato su `true`, non puoi mostrare il pulsante Guida sulla barra del titolo di `Form` e perderà la funzione di fare clic sul pulsante di aiuto per convertirlo per aiutare il cursore a poter fare clic sui controlli per mostra aiuto.

Puoi fare in modo che una voce di menu su `MenuStrip` come un pulsante di aiuto standard. Per fare ciò, aggiungi un `MenuStrip` al modulo e aggiungi un `ToolStripMenuItem`, quindi `MenuStrip` l'evento `Click` dell'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: se vuoi farlo usando un `Button`, devi anche impostare `button1.Capture = false;` prima di inviare il messaggio. Ma non è necessario per un `ToolStripMenuItem`.

Quindi quando fai clic sul menu della guida, il cursore verrà cambiato in ? cursore e agirà come quando si fa clic sul pulsante Guida standard:



Gestione di HelpButtonErrore evento di modulo

È possibile rilevare quando un utente fa clic su un `HelpButton` sulla barra del titolo del modulo gestendo `HelpButtonClicked`. Puoi lasciare che l'evento continui o cancellarlo impostando la proprietà `Cancel` dei suoi argomenti di evento su `true`.

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

Leggi Aiuta l'integrazione online: <https://riptutorial.com/it/winforms/topic/3285/aiuta-l-integrazione>

Capitolo 3: Associazione dati

Parametri

Discussione	Descrizione
nome della proprietà	Il nome della proprietà di controllo da associare.
fonte di dati	Un oggetto che rappresenta l'origine dati.
DataMember	La proprietà o l'elenco da associare a.
formattingEnabled	Determina se i dati visualizzati devono essere formattati.
UpdateMode	L'origine dati viene aggiornata quando la proprietà del controllo viene convalidata (impostazione predefinita) o immediatamente quando la proprietà è stata modificata
NullValue	Quando l'origine dati ha questo valore, la proprietà associata è impostata su DBNull.
formatString	Uno o più caratteri identificatori di formato che indicano come deve essere visualizzato un valore
FormatInfo	Un'implementazione di IFormatProvider per sovrascrivere il comportamento di formattazione predefinito.

Osservazioni

Vedere <https://msdn.microsoft.com/en-us/library/ef2xyb33.aspx> L' associazione dei dati funziona solo con le proprietà, mai con i campi!

Examples

Controllo dei collegamenti agli oggetti dati

Ogni controllo ha una proprietà `DataBindings` che è una lista di oggetti `System.Windows.Forms.Binding`. Il metodo `Add()` - ha alcuni overload che ti permettono di legare facilmente alla proprietà di un oggetto:

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

Nota, che il legame fondamentale significa iscriversi agli altri cambiamento di evento. Il

codice sopra sottoscrive il changeevent di dataObj.MyProperty e adatta textBox.Text quando cambia. E viceversa si abbona a textBox.TextChanged e adatta dataObj.MyProperty quando cambia.

Leggi Associazione dati online: <https://riptutorial.com/it/winforms/topic/7362/associazione-dati>

Capitolo 4: Casella di testo

Examples

Completamento automatico da una raccolta di stringhe

```
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);
```

Ciò si **completterà automaticamente** come l'utente prova a digitare **G** o **L**.

`AutoCompleteMode.SuggestAppend` visualizzerà entrambi un elenco di valori suggeriti e digiterà automaticamente la prima corrispondenza, `Append only` e `Suggest only` sono disponibili.

Consenti solo cifre nel testo

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

Ciò consentirà solo l'uso di cifre e caratteri di controllo nel `TextBox`, altre combinazioni sono possibili utilizzando lo stesso approccio di impostare la proprietà `Handle` su `true` per bloccare il testo.

L'utente può ancora copiare / incollare caratteri indesiderati, quindi dovrebbe essere presente un controllo aggiuntivo su `TextChanged` per pulire l'input:

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

In questo esempio viene utilizzata **un'espressione regolare** per filtrare il testo.

NumericUpDown dovrebbe essere preferito per i numeri quando possibile.

Come scorrere fino alla fine

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

Applicando lo stesso principio, `SelectionStart` può essere impostato su 0 per scorrere verso l'alto o verso un numero specifico per passare a un carattere specifico.

Aggiunta di un segnaposto alla casella di testo

Questo codice posiziona il testo del *suggerimento* al caricamento del modulo e lo manipola come segue:

C

```
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
```

Leggi Casella di testo online: <https://riptutorial.com/it/winforms/topic/4674/casella-di-testo>

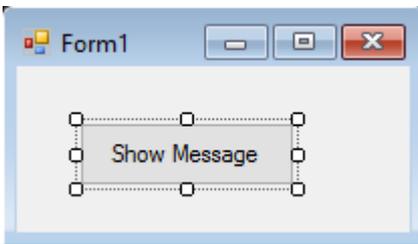
Capitolo 5: Controlli di base

Examples

Pulsante

I pulsanti sono uno dei controlli più semplici e vengono principalmente utilizzati per eseguire codice quando l'utente lo desidera.

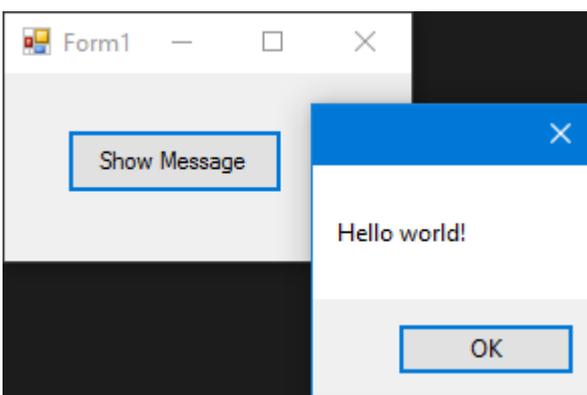
Qui abbiamo un caso davvero semplice, mostra una finestra di messaggio quando fai clic su un pulsante. Aggiungiamo un pulsante a un modulo, lo `cmdShowMessage` come usato nel codice per identificare l'oggetto e impostare il testo dei pulsanti su Mostra messaggio.



Abbiamo solo bisogno di fare doppio clic sul pulsante sul visual designer e Visual Studio genererà il codice per l'evento click. Ora abbiamo solo bisogno di aggiungere il codice per MessageBox lì:

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

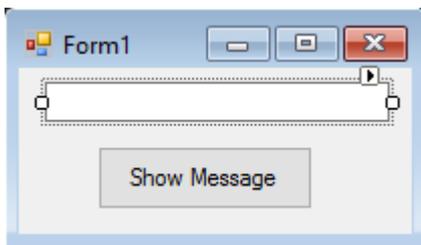
Se eseguiamo il programma ora e clicchiamo sul pulsante vedremo apparire il messaggio:



Casella di testo

I TextBox consentono all'utente di inserire dati nel programma.

Stiamo andando a modificare il modulo e aggiungere una casella di testo in modo che la messagebox ci mostri il messaggio che l'utente desidera. Ora la nostra forma assomiglia a:

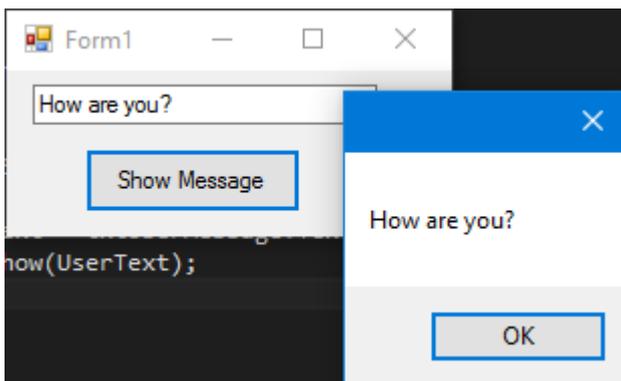


E poi modifica l'evento click del pulsante per usare il testo della textbox:

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

Come puoi vedere stiamo usando la proprietà `.Text` della Textbox che è il testo contenuto nella textbox.

Se eseguiamo il programma, saremo in grado di scrivere nella casella di testo. Quando clicchiamo sul pulsante, `MessageBox` mostrerà il testo che abbiamo scritto:

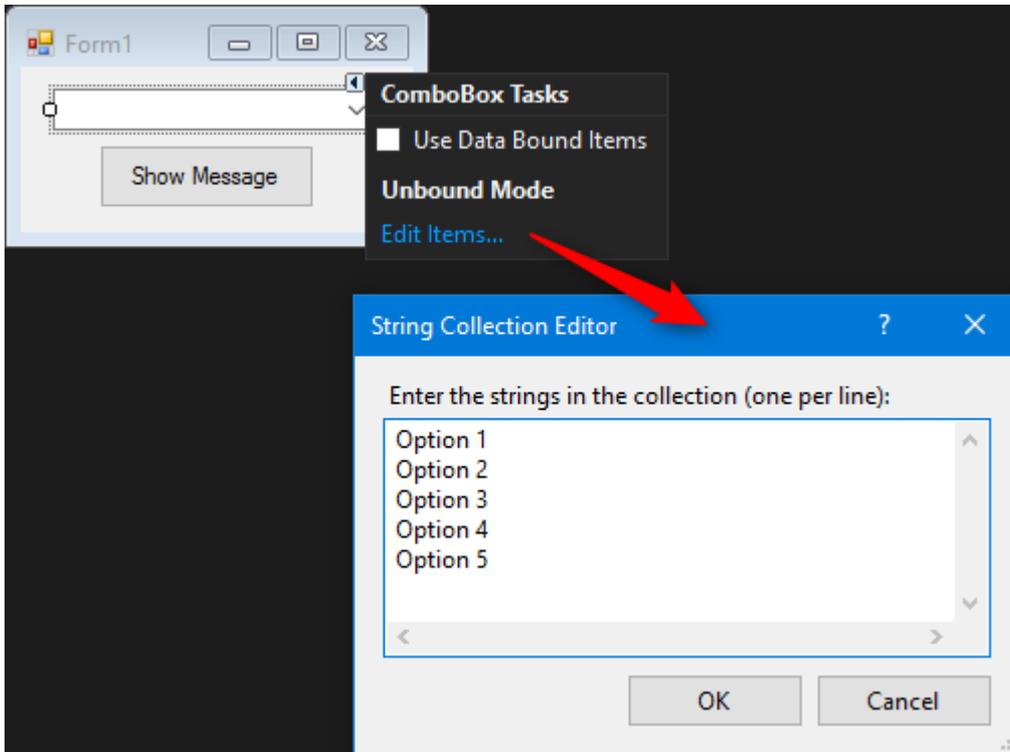


Casella combinata

I `ComboBox` consentono all'utente di scegliere una delle varie opzioni fornite dallo sviluppatore.

Stiamo andando a modificare il modulo e aggiungere una casella combinata in modo che la `messagebox` ci mostri il messaggio che l'utente desidera da un elenco che forniremo.

Dopo aver aggiunto la combinazione al modulo, ora aggiungiamo un elenco di opzioni alla `combo`. Per fare ciò dobbiamo modificare la proprietà `Items` :

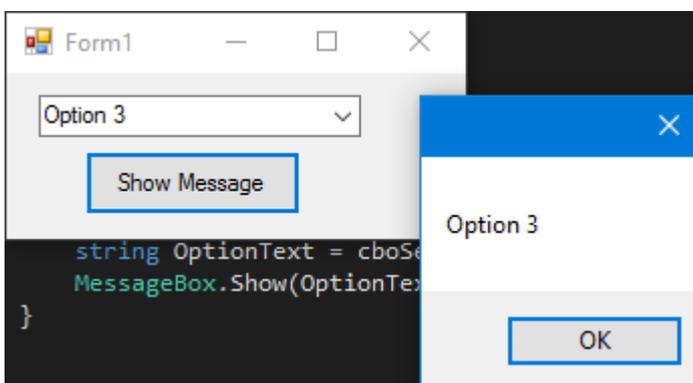


Ora dobbiamo modificare il codice dell'evento click:

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

Come puoi vedere usiamo la proprietà `SelectedItem`, contiene l'oggetto dell'opzione selezionata. Poiché abbiamo bisogno di una stringa da mostrare e il compilatore non sa se l'oggetto è o non è una stringa, dobbiamo usare il metodo `ToString()`.

Se eseguiremo il programma, saremo in grado di scegliere l'opzione che preferiamo e quando clicchiamo sul pulsante la finestra del messaggio lo mostrerà:



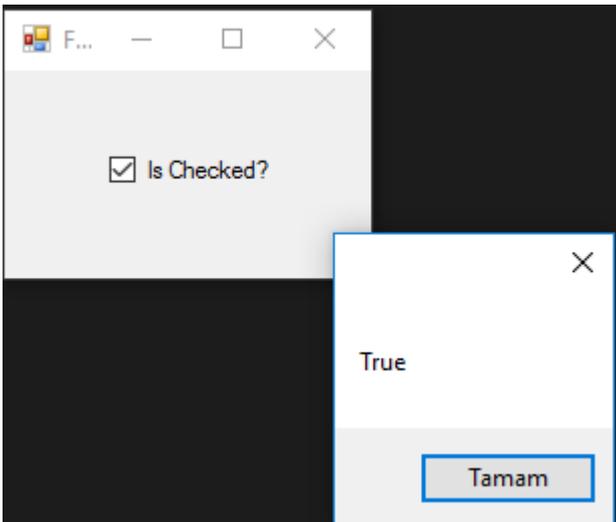
Per ricevere una notifica quando un utente seleziona un elemento dalla casella combinata, utilizzare l'evento `SelectionChangeCommitted`. Potremmo utilizzare l'evento `SelectedIndexChanged`, ma questo viene sollevato anche quando modifichiamo a livello di codice l'elemento selezionato nella casella combinata.

CheckBox

Checkbox è un controllo che consente all'utente di ottenere valori `boolean` dall'utente per una domanda specificata come "**Stai bene?**".

Ha un evento chiamato `CheckedChanged`, che si verifica ogni volta che la proprietà di `check` viene modificata.

Ecco un `CheckBox` che ha una domanda "**È verificato?**".



Abbiamo ricevuto questo `MessageBox` dall'evento `CheckedChanged`,

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

Se `CheckBox` è selezionato -> `IsChecked` variabile `IsChecked` sarà `true`.

Se `CheckBox` non è selezionato -> `IsChecked` variabile `IsChecked` sarà `false`.

ListBox

`Listbox` è un controllo che può contenere una raccolta di oggetti. `Listbox` è simile a `Combobox` ma in `Combobox`; Solo gli elementi selezionati sono visibili all'utente. In `Listbox`; tutti gli articoli sono visibili all'utente.

Come aggiungere elementi a `ListBox`?

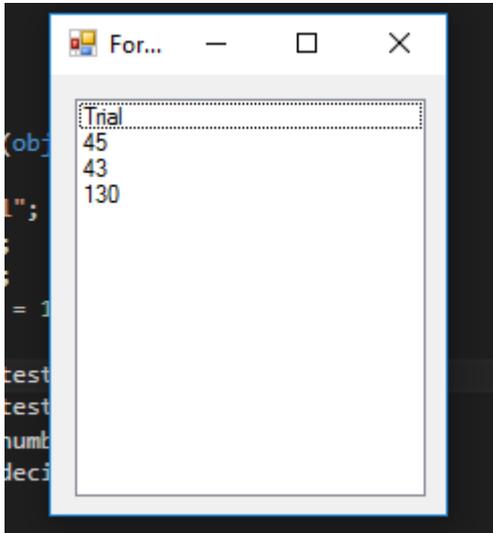
```
private void Form3_Load(object sender, EventArgs e)
{
    string test = "Trial";
    string test2 = "45";
    int numberTest = 43;
    decimal decimalTest = 130;
```

```

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

```

Uscita ;



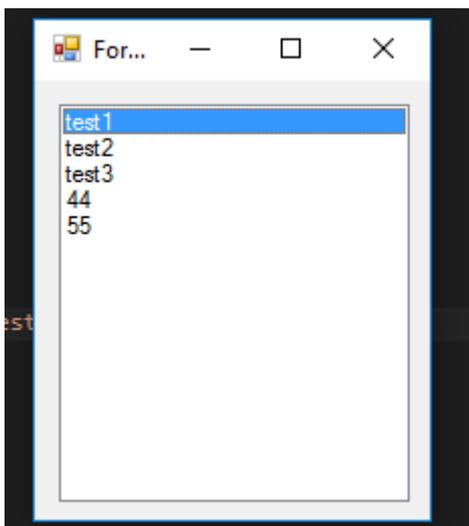
Oppure possono essere fornite `datasources` dati

```

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

```

Produzione;



```

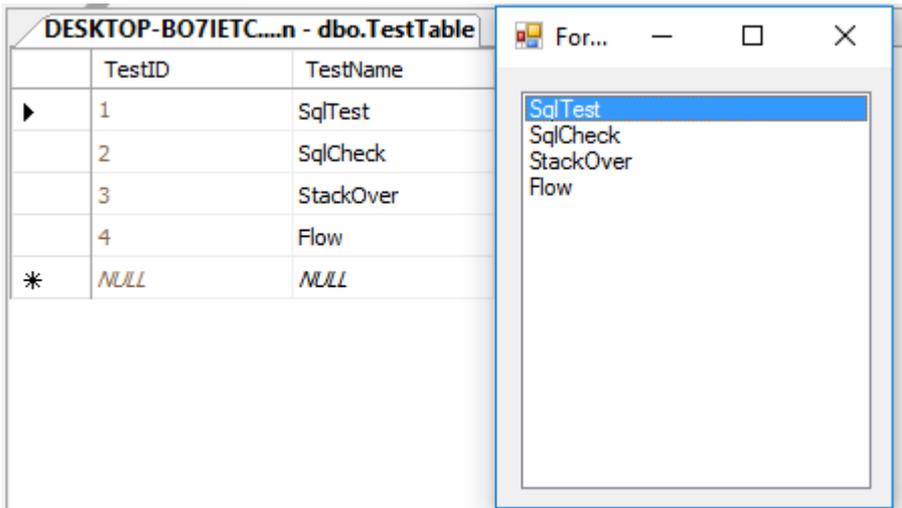
private void Form3_Load(object sender, EventArgs e)
{
    SqlConnection Connection = new
    SqlConnection("Server=serverName;Database=db;Trusted_Connection=True;"); //Connetion to MS-

```

SQL (RDBMS)

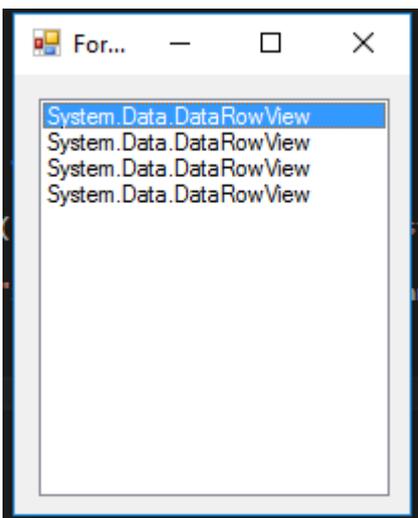
```
Connection.Open(); //Connection open
SqlDataAdapter Adapter = new SqlDataAdapter("Select * From TestTable",
Connection); // Get all records from TestTable.
DataTable DT = new DataTable();
Adapter.Fill(DT); // Fill records to DataTable.
listBox1.DataSource = DT; // DataTable is the datasource.
listBox1.ValueMember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

L' uscita corretta ;



È necessario ValueMember dati sql esterna a listbox, ValueMember e DisplayMember

Se **NON** sarà così,



Eventi utili;

SelectedIndex_Changed;

Definire una lista per dare origine dati

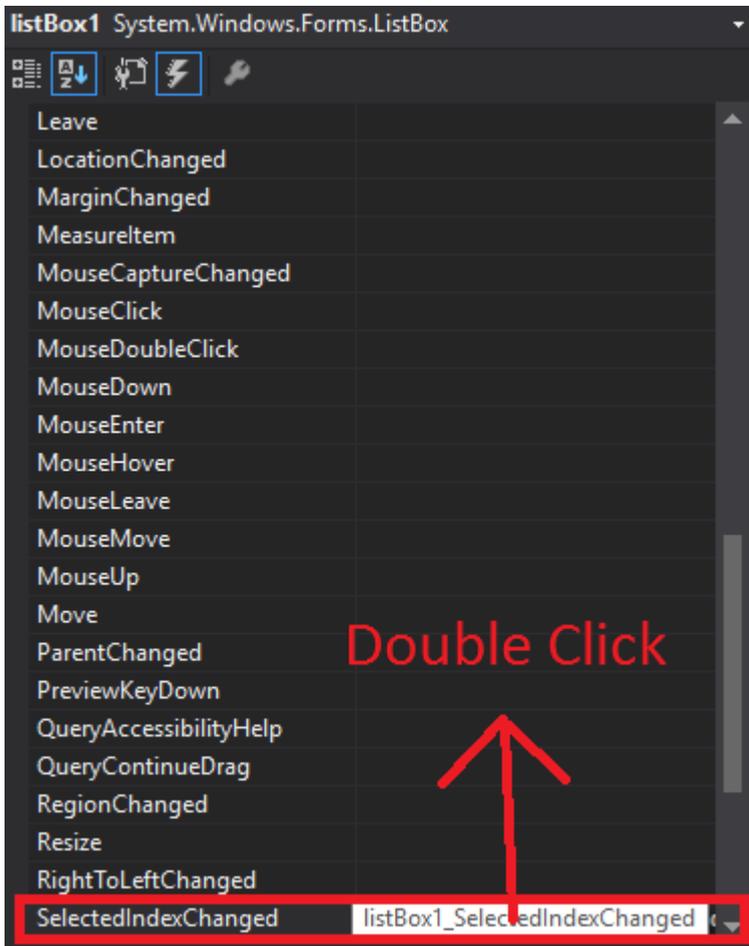
```
private void Form3_Load(object sender, EventArgs e)
```

```

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

```

Alla progettazione del modulo seleziona `Listbox` e premi F4 o fai clic con il pulsante destro sull'icona lightning.



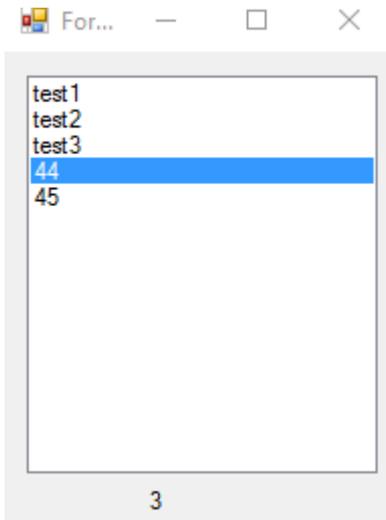
Visual Studio genererà `listBox1_SelectedIndexChanged` su codebehind.

```

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

```

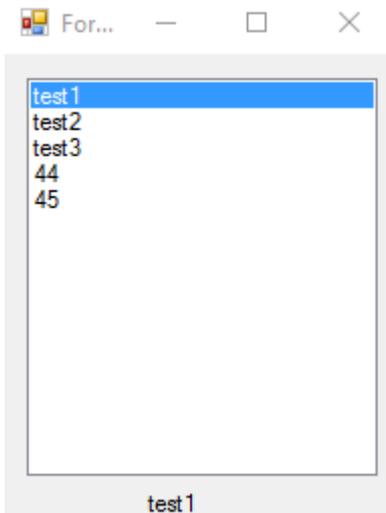
Risultato di `SelectedIndex_Changed` ; (l'etichetta in basso mostrerà l'indice di ciascun articolo selezionato)



SelectedValue_Changed; (L'origine dati è la stessa in alto e puoi generare questo evento come SelectedIndex_Changed)

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

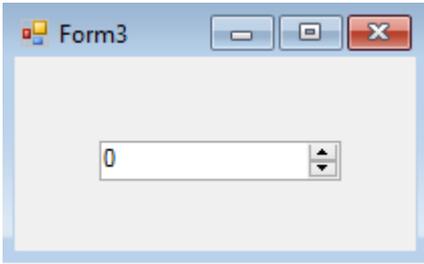
Uscita ;



NumericUpDown

NumericUpDown è un controllo simile a TextBox. Questo controllo consente all'utente di visualizzare / selezionare il numero da un intervallo. Le frecce Su e Giù stanno aggiornando il valore della casella di testo.

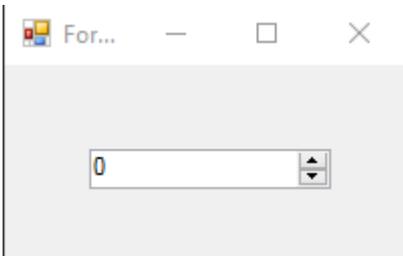
Il controllo assomiglia;



In `Form_Load` intervallo può essere impostato.

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

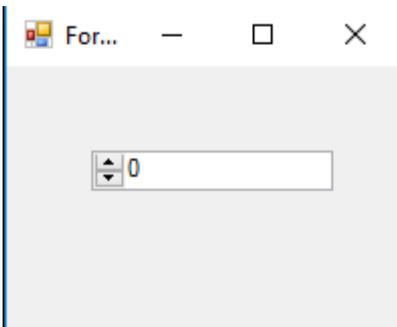
Produzione;



UpDownAlign imposterà la posizione delle frecce;

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

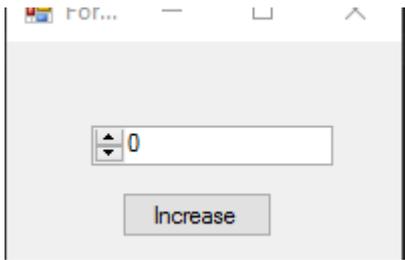
Produzione;



Metodo `UpButton()` aumenta il numero del controllo. (può essere chiamato da qualsiasi luogo. Ho usato un `button` per chiamarlo.)

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

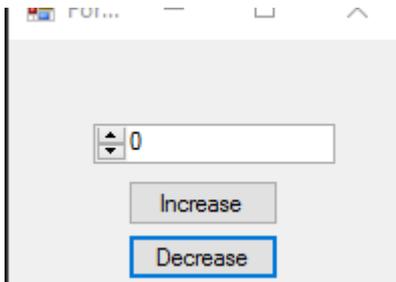
**Produzione



Metodo `DownButton()` diminuisce il numero del controllo. (può essere chiamato da qualsiasi luogo. Ho usato un `button` per chiamarlo di nuovo.)

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

Produzione;



Eventi utili

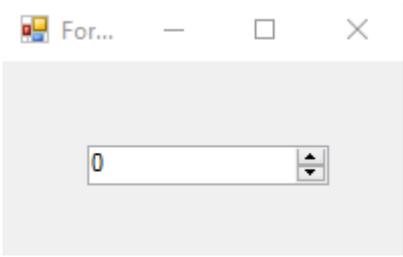
ValueChanged;

Quell'evento funzionerà quando si fa clic sulla freccia Su o Giù.

```
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.
    }
}
```

```
}  
}
```

Uscita ;



Leggi Controlli di base online: <https://riptutorial.com/it/winforms/topic/5816/controlli-di-base>

Capitolo 6: Controlli ereditari

Osservazioni

I controlli sono derivati esattamente nello stesso modo delle altre classi. L'unica cosa da tenere a mente è l'override degli eventi: solitamente è consigliabile assicurarsi di chiamare il gestore di eventi di base dopo il proprio. La mia regola generale: in caso di dubbio, chiama l'evento di base.

Examples

Impostazioni a livello di applicazione

Una lettura veloce della maggior parte dei siti degli sviluppatori rivelerà che WinForms è considerato inferiore a WPF. Uno dei motivi più frequentemente citati è la supposta difficoltà nel fare ampie modifiche dell'applicazione al "look-and-feel" di un'intera applicazione.

In realtà è sorprendentemente facile produrre un'applicazione in WinForms che sia facilmente configurabile sia in fase di progettazione che in fase di esecuzione, se si evita semplicemente l'uso dei controlli standard e si ricava il proprio da essi.

Prendi il TextBox come esempio. È difficile immaginare un'applicazione Windows che non richiede l'uso di un controllo TextBox in un determinato momento. Pertanto, avere il proprio TextBox avrà sempre senso. Prendi il seguente esempio:

```
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 delle cose che gli utenti trovano più utili in un modulo di inserimento dati, con molte caselle di input, è di avere il colore di sfondo della scatola con il cambio di messa a fuoco. Visibilmente è più facile da vedere, rispetto a un cursore verticale lampeggiante standard. Il codice sopra fornisce un TextBox che fa esattamente questo.

Nel processo utilizza le proprietà statiche di una classe statica. Dò sotto un estratto dal mio:

```
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);
    }
}
}
}
}

```

Questa classe utilizza il registro di Windows per mantenere le proprietà, ma se lo preferisci puoi utilizzare un database o un file di impostazioni. Il vantaggio di utilizzare una classe statica in questo modo è che le modifiche a livello di applicazione possono essere apportate non solo in fase di progettazione, ma anche dall'utente in fase di esecuzione. Includo sempre un modulo nelle mie applicazioni che consente all'utente di modificare i valori preferiti. La funzione di salvataggio non solo salva nel registro (o nel database, ecc.), Ma anche in fase di esecuzione modifica le proprietà nella classe statica. Si noti che le proprietà statiche di una classe statica non sono costanti; in questo senso possono essere considerati come variabili a livello di applicazione. Ciò significa che qualsiasi modulo aperto successivamente alle modifiche salvate sarà immediatamente interessato dalle eventuali modifiche salvate.

Potrai facilmente essere in grado di pensare ad altre proprietà dell'applicazione che vorresti essere configurabili allo stesso modo. I caratteri sono un altro ottimo esempio.

NumberBox

Spesso vorrete avere una casella di input che accetta solo numeri. Ancora una volta derivando dai controlli standard questo si ottiene facilmente, ad esempio:

```

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;
    }
}

```

Oltre a limitare l'input ai numeri, questa classe ha alcune caratteristiche speciali. Espone una proprietà Value per rappresentare il doppio valore del numero, formatta il testo, facoltativamente con migliaia di separatori e fornisce l'immissione a breve termine di numeri grandi: 10M si espande in lasciare a 10.000.000,00 (il numero di posizioni decimali come proprietà). Per brevità, i decimali e i mille separatori sono stati codificati a fondo. In un sistema di produzione, anche queste sono preferenze dell'utente.

Leggi Controlli ereditari online: <https://riptutorial.com/it/winforms/topic/6476/controlli-ereditari>

Capitolo 7: Mostrare un modulo

introduzione

Questo argomento spiega come funziona il motore WinForms per visualizzare i moduli e il modo in cui si controlla la loro durata.

Examples

Mostra una forma modale o modale

Dopo aver definito la struttura del modulo con il designer WinForms, è possibile visualizzare i moduli in codice con due metodi diversi.

- Metodo: una forma non modale

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

- Metodo: una finestra di dialogo modale

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

I due metodi hanno una distinzione molto importante. Il primo metodo (quello non modale) mostra il tuo modulo e quindi ritorna immediatamente senza attendere la chiusura del modulo appena aperto. Quindi il tuo codice continua con quello che segue la chiamata Show. Il secondo metodo invece (quello modale) apre il modulo e blocca qualsiasi attività sull'intera applicazione fino a quando non si chiude il modulo tramite il pulsante di chiusura o con alcuni pulsanti opportunamente configurati per chiudere il modulo

Chiusura di un modulo non modale

Viene utilizzato un modulo non modale (di solito) quando è necessario mostrare qualcosa in modo permanente accanto alla schermata principale dell'applicazione (si pensi a una legenda o una vista su un flusso di dati proveniente in modo asincrono da un dispositivo o una finestra secondaria MDI).

Ma una forma non modale pone una sfida unica quando vuoi chiuderla. Come recuperare l'istanza e chiamare il metodo Close in quell'istanza?

È possibile mantenere una variabile globale che fa riferimento all'istanza che si desidera chiudere.

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

Ma possiamo anche scegliere di utilizzare la raccolta `Application.OpenForms` in cui il modulo modulo memorizza tutte le istanze del modulo create e ancora aperte.

È possibile recuperare questa particolare istanza da questa raccolta e chiamare il metodo `Close`

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

Chiusura di una forma modale

Quando un modulo viene mostrato utilizzando il metodo `ShowDialog`, è necessario impostare la proprietà `DialogResult` del `DialogResult` per chiudere il modulo. Questa proprietà può essere impostata utilizzando l' [enumerazione](#) chiamata anche `DialogResult`.

Per chiudere un modulo, è sufficiente impostare la proprietà `DialogResult` del `DialogResult` (su qualsiasi valore in `DialogResult.None`) in un gestore di eventi. Quando il tuo codice esce dal gestore di eventi, il motore WinForm nasconde il modulo e il codice che segue la chiamata del metodo `ShowDialog` iniziale continuerà l'esecuzione.

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

Il codice chiamante può acquisire il valore restituito da `ShowDialog` per determinare quale pulsante l'utente ha fatto clic nel modulo. Quando viene visualizzato utilizzando `ShowDialog()`, il modulo non viene eliminato automaticamente (poiché era semplicemente nascosto e non chiuso), quindi è importante utilizzare un blocco `using` per assicurarsi che il modulo sia stato eliminato.

Di seguito è riportato un esempio di controllo del risultato dell'utilizzo di `OpenFileDialog` incorporato, controllo del risultato e accesso a una proprietà dalla finestra di dialogo prima di eliminarlo.

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

È inoltre possibile impostare la proprietà `DialogResult` su un pulsante. Facendo clic su questo pulsante si imposta la proprietà `DialogResult` sul modulo sul valore associato al pulsante. Ciò consente di chiudere il modulo senza aggiungere un gestore di eventi per impostare `DialogResult` nel codice.

Ad esempio, se si aggiunge un pulsante OK al modulo e si imposta la proprietà su `DialogResult.OK`

il modulo si chiude automaticamente quando si preme quel pulsante e il codice chiamante riceve un `DialogResult.OK` in risposta alla chiamata del metodo `ShowDialog()` .

Leggi **Mostrare un modulo online**: <https://riptutorial.com/it/winforms/topic/8768/mostrare-un-modulo>

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con Winforms	4444 , Bjørn-Roger Kringsjå , Chris Shao , Cody Gray , Community , Reza Aghaei
2	Aiuta l'integrazione	help-info.de , Reza Aghaei
3	Associazione dati	Kai Thoma
4	Casella di testo	gplumb , Jones Joseph , Stefano d'Antonio
5	Controlli di base	Aimnox , Berkay , help-info.de , Jeff Bridgman
6	Controlli ereditari	Balagurunathan Marimuthu
7	Mostrare un modulo	Cody Gray , Jeff Bridgman , Steve