## FREE eBook

# LEARNING winforms

Free unaffiliated eBook created from **Stack Overflow contributors.** 

## #winforms

## **Table of Contents**

About1
Chapter 1: Getting started with winforms
Remarks
See also:
Examples
Creating a Simple WinForms Application using Visual Studio2
Create Windows Forms Project
Add Controls to the Form
Write Code4
Run and Test5
Creating a Simple C# WinForms Application using a Text Editor5
Creating a Simple VB.NET WinForms Application using a Text Editor
Chapter 2: Basic controls 9
Examples
Button
TextBox
ComboBox10
CheckBox11
ListBox12
NumericUpDown
Useful Events
Chapter 3: Databinding
Parameters
Remarks
Examples
Binding controls to data objects
Chapter 4: Help Integration
Remarks
HelpProvider Component
Help Class

HelpRequested Event	21
Help Button of Form	21
Help button of MessgeBox and CommonDialogs	21
ToolTip Component	22
Examples	
Show help file	22
Show Help for MessageBox	22
Show a CHM file and navigate to a keyword (index)	23
Show a CHM file and navigate to a topic	
Show a CHM file and navigate first help page in table of contents	23
Open default browser and navigate to a URL	23
Perform custom action on when pressing Help Button or F1 Key	
Show Help for CommonDialogs	24
Handling HelpRequested event of Controls and Form	25
Show Help using Help class	26
Show Help pop-up window	
Show CHM Help file	
Show Help Table of Content	
Show Help for specific Keyword (index)	
Show Help for specific Topic	27
Show Url	27
Show Help Button on Titlebar of Form	27
Create custom Help button which acts like standard Form HelpButton	27
	00
Handling HelpButtonClicked event of Form	
Handling HelpButtonClicked event of Form         Chapter 5: Inheriting Controls	
Handling HelpButtonClicked event of Form	

Show a modeless or a modal form
Closing a modeless form
Closing a modal form4
Chapter 7: TextBox 42
Examples
Auto completion from a collection of strings4
Allow only digits in the text
How to scroll to the end
Adding a Placeholder to textbox4
Gredits



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

## **Chapter 1: Getting started with winforms**

## Remarks

**Windows Forms** ("WinForms" for short) is a GUI class library included with the .NET Framework. It is a sophisticated object-oriented wrapper around the Win32 API, allowing the development of Windows desktop and mobile applications that target the .NET Framework.

WinForms is primarily event-driven. An application consists of multiple *forms* (displayed as windows on the screen), which contain *controls* (labels, buttons, textboxes, lists, etc.) that the user interacts with directly. In response to user interaction, these controls raise events that can be handled by the program to perform tasks.

Like in Windows, everything in WinForms is a control, which is itself a type of window. The base Control class provides basic functionality, including properties for setting text, location, size, and color, as well as a common set of events that can be handled. All controls derive from the Control class, adding additional features. Some controls can host other controls, either for reusability (Form , UserControl) or layout (TableLayoutPanel, FlowLayoutPanel).

WinForms has been supported since the original version of the .NET Framework (v1.0), and is still available in modern versions (v4.5). However, it is no longer under active development, and no new features are being added. According to 9 Microsoft developers at the Build 2014 conference:

Windows Forms is continuing to be supported, but in maintenance mode. They will fix bugs as they are discovered, but new functionality is off the table.

The cross-platform, open-source Mono library provides a basic implementation of Windows Forms, supporting all of the features that Microsoft's implementation did as of .NET 2.0. However, WinForms is not actively developed on Mono and a complete implementation is considered impossible, given how inextricably linked the framework is with the native Windows API (which is unavailable in other platforms).

See also:

Windows Forms documentation on MSDN

## Examples

**Creating a Simple WinForms Application using Visual Studio** 

This example will show you how to create a Windows Forms Application project in Visual Studio.

## **Create Windows Forms Project**

1. Start Visual Studio.

- 2. On the **File** menu, point to **New**, and then select **Project**. The **New Project** dialog box appears.
- 3. In the Installed Templates pane, select "Visual C#" or "Visual Basic".
- 4. Above the middle pane, you can select the target framework from the drop-down list.
- 5. In the middle pane, select the Windows Forms Application template.
- 6. In the **Name** text box, type a name for the project.
- 7. In the **Location** text box, choose a folder to save the project.
- 8. Click **OK**.
- 9. The Windows Forms Designer opens and displays Form1 of the project.

## Add Controls to the Form

- 1. From the **Toolbox** palette, drag a **Button** control onto the form.
- 2. Click the button to select it. In the Properties window, set the Text property to Say Hello.



## Write Code

- 1. Double-click the button to add an event handler for the Click event. The Code Editor will open with the insertion point placed within the event handler function.
- 2. Type the following code:

C#

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

#### **VB.NET**

MessageBox.Show("Hello, World!")

### **Run and Test**

1. Press  $_{\text{F5}}$  to run the application.

•	Form1	_ □	×
	Sav	Hello	
	Say	IBIID	

2. When your application is running, click the button to see the "Hello, World!" message.

		×
Hello	World!	
[	OK	

3. Close the form to return to Visual Studio.

Creating a Simple C# WinForms Application using a Text Editor

1. Open a text editor (like Notepad), and type the code below:

```
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. Save the file to a path you have read/write access to. It is conventional to name the file after the class that it contains—for example, x:\MainForm.cs.
- 3. Run the C# compiler from the command line, passing the path to the code file as an argument:

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

*Note:* To use a version of the C# compiler for other .NET framework versions, take a look in the path, <code>%WINDIR%\Microsoft.NET</code> and modify the example above accordingly. For more information on compiling C# applications, see Compile and run your first C# program.

 After compilation has completed, an application called MainForm.exe will be created in the same directory as your code file. You can run this application either from the command line or by double-clicking on it in Explorer.

Creating a Simple VB.NET WinForms Application using a Text Editor

1. Open a text editor (like Notepad), and type the code below:

```
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. Save the file to a path you have read/write access to. It is conventional to name the file after the class that it contains—for example, x:\MainForm.vb.
- 3. Run the VB.NET compiler from the command line, passing the path to the code file as an argument:

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

*Note:* To use a version of the VB.NET compiler for other .NET framework versions, take a look in the path <code>%WINDIR%\Microsoft.NET</code> and modify the example above accordingly. For more information on compiling VB.NET applications, see Hello World.

4. After compilation has completed, an application called MainForm.exe will be created in the same directory as your code file. You can run this application either from the command line or by double-clicking on it in Explorer.

Read Getting started with winforms online: https://riptutorial.com/winforms/topic/1018/gettingstarted-with-winforms

## **Chapter 2: Basic controls**

## Examples

#### **Button**

Buttons are one of the simplest controls and mostly used for executing some code when the user wants.

Here we have a really simple case, show a Message box when a button is clicked. We add a button to a form, name it cmdShowMessage as used in code to identify the object and set the buttons text to Show Message.



We just need to double click the button on the visual designer and Visual Studio will generate the code for the click Event. Now we just need to add the code for the MessageBox there:



If we run the program now and click the button we'll see the message appearing:



#### **TextBox**

TextBoxes allow the user to input data into the program.

We are going to modify the form and add a textbox so the messagebox show us the message that the user wants. Now our form looks like:

🖳 Form1 📃 🖃 🗾
Show Message

And then modify the button click event to use the text of the textbox:

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

As you can see we are using the .Text property of the Textbox that is the text contained in the texbox.

If we run the program, we will be able to write in the textbox. When we click the button the MessageBox will show the text that we have wrote:



#### ComboBox

ComboBoxes allow the user to choose one of various options provided by the developer.

We are going to modify the form and add a combobox so the messagebox show us the message that the user want from a list that we will provide.

After adding the combo to the form we now add a list of options to the combo. To do so we need to modify the Items property:

Form1 🗖 🖬 💈	E Contraction of the second		
	ComboBox Tasks		
	Use Data Bound Items		
Show Message	Unbound Mode		
	Edit Items		
	String Collection Editor		? ×
	Enter the strings in the co	llection (one pe	er line):
	Option 1		~
	Option 2 Option 3		
	Option 4		
	Option 5		
	<		>
		ОК	Cancel

Now we need to modify the code of the click event:

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

As you can see we use the *SelectedItem* property, it contains the object of the selected option. Since we need a string to show and the compiler does not know if the object is or isn't a string, we need to use the *ToString()* method.

If we run the program we'll be able to choose the option that we prefer and when we click the button the message box will show it:



To be notified when an user selects an item from the combobox, use the selectionChangeCommitted
event. We could use the selectedIndexChanged
event, but this is also raised when we
programmatically change the select item in the combobox.

#### CheckBox

https://riptutorial.com/

Checkbox is a control that allow user to get boolean values from user for a spesific question like "Are you ok?".

Has a event called CheckedChanged, which occurs whenever the check property is changed.

Here is a CheckBox that has a question "Is Checked?".



We got this MessageBox from CheckedChanged event,

If CheckBox is checked -> IsChecked variable will be true.

If CheckBox is not checked -> IsChecked variable will be false.

#### **ListBox**

Listbox is a control that can contains collection of objects. Listbox is similar to Combobox but in Combobox; Only selected items are visible to user. In Listbox; all items are visible to user.

#### How to add items to 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);
}
```

#### Output;



Or datasources can be given,

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

#### Output;



```
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";
}
```

#### The proper output;

D	ESKTOP-BO7IE	TCn - dbo.TestTable	6	🖶 For	_	Х			
	TestID	TestName							
•	1	SqlTest		SolTest SolCheck StackOver	SalTest	SalTest			
	2	SqlCheck							
	3	StackOver		Flow					
	4	Flow							
*	NULL	NULL							

Giving an external sql datasource to listbox requires, valueMember and DisplayMember

If **NOT** it will looks like this,

🖶 For	_		×
System.Da System.Da System.Da System.Da	ta Data ta.Data ta.Data ta.Data	RowView RowView RowView RowView	

Useful events;

#### SelectedIndex\_Changed;

Define a list to give datasource

```
private void Form3_Load(object sender, EventArgs e)
{
    List<string> DataList = new List<string> {"test1", "test2", "test3", "44",
"45" };
    listBox1.DataSource = TestList;
    }
```

At the form's design select Listbox and press F4 or at right side click on lightining icon.



Visual studio will generate <code>listBox1\_SelectedIndexChanged</code> to codebehind.

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

Result of selectedIndex\_Changed; (label at the bottom will show the index of each selected item)



**SelectedValue\_Changed;** (The datasource is same as at the top and you can generate this event like SelectedIndex\_Changed)

#### Output;



#### NumericUpDown

NumericUpDown is control that looks like TextBox. This control allow user to display/select number from a range. Up and Down arrows are updating the textbox value.

Control look like;



In  ${\tt Form\_Load}$  range can be set.

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

#### Output;



#### UpDownAlign will set the position of arrows;

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

#### Output;



UpButton() Method increase the number of the control. (can be called from anywhere. I used a button to call it.)

\*\*Output



DownButton() Method decrease the number of the control. (can be called from anywhere. I used a button to call it again.)

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

#### Output;



## **Useful Events**

#### ValueChanged;

That event will work when Up or Down arrow clicked.

```
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.
            }
       }
```

#### Output;



Read Basic controls online: https://riptutorial.com/winforms/topic/5816/basic-controls

## **Chapter 3: Databinding**

## Parameters

Argument	Description
propertyName	The name of the control property to bind.
dataSource	An Object representing the data source.
dataMember	The property or list to bind to.
formattingEnabled	Determines, whether the displayed data should be formatted.
updateMode	Data source is updated when the control property is validated (default), or immediately when the property has changed
nullValue	When the data source has this value, the bound property is set to DBNull.
formatString	One or more format specifier characters that indicate how a value is to be displayed
formatInfo	An implementation of IFormatProvider to override default formatting behavior.

## Remarks

See https://msdn.microsoft.com/en-us/library/ef2xyb33.aspx Databinding only works with properties, never with fields!

## Examples

Binding controls to data objects

Each control has a property DataBindings which is a list of System.Windows.Forms.Binding objects. The Add()-method has some overloads which enables you easily binding to the property of an object:

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

Note, that binding basically means subscribing to each others changeevent. The code above subscribes to changeevent of dataObj.MyProperty and adapts textBox.Text when it changes. And vice versa it subscribes to textBox.TextChanged and adapts dataObj.MyProperty when it changes.

Read Databinding online: https://riptutorial.com/winforms/topic/7362/databinding

## **Chapter 4: Help Integration**

## Remarks

You can provide help for Forms and Controls in a Windows Forms Applications in different ways. You can show a pop-up help, open a CHM file or a URL. You can show context-sensitive help for Forms, Controls and dialogs.

## HelpProvider Component

You can setup a HelpProvider component to provide context sensitive help for component. This way when the user press F1 key or Help button of form, you can automatically:

- Show a context sensitive help pop-up for controls
- Open a CHM file based on context (Show table of content, Show a keyword or index, show a topic)
- Navigate to a URL using default browser

## **Help Class**

You can use Help class in code, to provide these kinds of help:

- Show a help pop-up for a control
- Open a CHM file based on context (Show table of content, Show a keyword or index, show a topic)
- Navigate to a URL using default browser

## **HelpRequested Event**

You can handle HelpRequested event of Control objects or Form to perform custom actions when the user press F1 or click on Help button of form.

## Help Button of Form

You can setup the Form to show Help button on title-bar. This way, if the user click on Help button, the cursor will change to a ? cursor and after click on any point, any context-sensitive help associated with the control using HelpProvider will be show.

## Help button of MessgeBox and CommonDialogs

You can provide help for MessageBox, OpenFileDialog, SaveDialog and ColorDialog using Help button of the components.

## **ToolTip Component**

You can use  $T_{OOLTip}$  component to display some help text when the user points at controls. A  $T_{OOLTip}$  can be associated with any control.

#### Note

Using HelpProvider and Help class You can show compiled Help files (.chm) or HTML files in the HTML Help format. Compiled Help files provide a table of contents, an index, search capability, and keyword links in pages. Shortcuts work only in compiled Help files. You can generate HTML Help 1.x files by using HTML Help Workshop. For more information about HTML Help, see "HTML Help Workshop" and other HTML Help topics at Microsoft HTML Help.

## **Examples**

#### Show help file

The Help Class encapsulates the HTML Help 1.0 engine. You can use the Help object to show compiled Help files (.chm) or HTML files in the HTML Help format. Compiled Help files provide table of contents, index, search, and keyword links in pages. Shortcuts work only in compiled Help files. You can generate HTML Help 1.x files with a free tool from Microsft called HTML Help Workshop

An easy way to show a compiled help file in a second window:

#### C#

Help.ShowHelp(this, helpProviderMain.HelpNamespace);

#### **VB.NET**

Help.ShowHelp(Me, hlpProviderMain.HelpNamespace)

#### Show Help for MessageBox

You can provide help for message box in different ways. You can configure a MessageBox to show a Help button or not. Also you can configure MessageBox in a way that when the user requests for help by click on Help button or by pressing F1, it show a CHM file or navigate to a URL or perform a custom action. Here are some examples in this topic.

In all below examples, the  ${\tt MessageBox}$  would be like this:

	Titl	e	×
Some Messag	ge		
Yes	No	Cancel	Help

## Show a CHM file and navigate to a keyword (index)

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

### Show a CHM file and navigate to a topic

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

# Show a CHM file and navigate first help page in table of contents

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

### Open default browser and navigate to a URL

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

## Perform custom action on when pressing Help Button or F1 Key

In this case you should handle HelpRequested event of parent of MessageBox and perform custom operation:

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

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

Then you can show the MessageBox with Help button:

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

#### Or show it without Help button:

#### Show Help for CommonDialogs

You can provide help for OpenFileDialog, SaveFileDialog and ColorDialog. To do so set ShowHelp property of dialog to true and handle HelpRequest event for dialog:

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

#### Note

- The event will be raised only if you set showHelp to true.
- The event will be raised only by click on Help button and will not raise using F1 key.

In the image below you can see an OpenFileDialog with a Help button:



#### Handling HelpRequested event of Controls and Form

When a user press F1 on a control or click on Help button of form (?) and then clicks on a control the HelpRequested event will be raised.

You can handle this event to provide custom action when user requests help for controls or form.

The HelpRequested supports bubble up mechanism. It fires for your active control and if you don't handle the event and not set Handled property of its event arg to true, then it bubbles up to the parent control hierarchy up to form.

For example if you handle HelpRequested event of the form like below, then when you press F1 a message box will pop up and show name of active control, but for textBox1 it will show a different message:

```
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;
```

You can perform any other custom action like using navigating to a URL or showing a CHM file using Help class.

#### Show Help using Help class

}

You can use Help class in code, to provide these kinds of help:

- Show a help pop-up for a control
- Open a CHM file based on context (Show table of content, Show a keyword or index, show a topic)
- Navigate to a URL using default browser

#### Show Help pop-up window

You can use Help.ShowPopup to display a help pop-up window:

```
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));
}
```

It will show such help pop-up at your mouse pointer location:

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

### Show CHM Help file

You can use different overloads of Help.ShowHelp method, to show a CHM file and navigate to a keyword, a topic, index or table of content:

Show Help Table of Content

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

Show Help for specific Keyword (index)

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

#### Show Help for specific Topic

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

### Show Url

You can show any URL in default browser using showHelp method:

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

Show Help Button on Titlebar of Form

You can show a Help Button at title-bar of a Form. To do so, you should:

- 1. Set HelpButton property of form to true.
- 2. Set MinimizeBox and MaximizeBox to false.

Then a help button will appear on title-bar of Form:



Also when you click on Help button, the cursor will be changed to a ? cursor:



Then if you click on a Control or Form, the HelpRequested event will be raised and also if you have setup a HelpProvider, the help for the control will be shown using HelpProvider.

#### Create custom Help button which acts like standard Form HelpButton

If you have a Form with MinimizeBox and MaximizeBox set to true, then you can not show Help button on title-bar of Form and will lose the feature of click on help button to convert it to help cursor to be able to click on controls to show help. You can make a menu item on MenuStrip act like standard Help Button. To do so, add a MenuStrip to the form and add a ToolStripMenuItem to it, then handle click event of the item:

```
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);
}
```

**Note:** If you want to do it using a Button, you also need to set button1.Capture = false; before sending the message. But it's not necessary for a ToolStripMenuItem.

Then when you click on the help menu, the cursor will be changed to 2 cursor and will act like when you click on standard Help button:

•	Forn	n1 —	×
<u>F</u> ile	<u>E</u> dit	<u>T</u> ools	۲
		₿ ₽	

#### Handling HelpButtonClicked event of Form

You can detect when a user Clicked on a HelpButton on title-bar of form by handling HelpButtonClicked. You can let the event continue or cancel it by setting Cancel property of its event args to true.

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

Read Help Integration online: https://riptutorial.com/winforms/topic/3285/help-integration

## **Chapter 5: Inheriting Controls**

### Remarks

Controls are derived in exactly the same way as other classes. The only thing to be careful of is overriding events: it is usually advisable to make sure that you call the base event handler after your own. My own rule of thumb: if in doubt, call the base event.

### **Examples**

#### **Application wide Settings**

A quick read of most developer sites will reveal that WinForms is considered inferior to WPF. One of the most often cited reasons is the supposed difficulty in making application wide changes to the "look-and-feel" of an entire application.

In fact it is surprisingly easy to produce an application in WinForms that is easily configurable both at design-time and run-time, if you simply eschew the use of the standard controls and derive your own from them.

Take the TextBox as an example. It is hard to imagine a Windows application that does not call for the use of a TextBox at some stage or other. Therefore, having your own TextBox will always make sense. Take the following example:

```
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);
        }
    }
```

One of the things that users find most helpful in a data entry form, with many input boxes, is to have the background colour of the box with focus change. Visibly it is easier to see, than a standard blinking vertical cursor. The above code provides a TextBox that does precisely that.

In the process it makes use of the static properties of a static class. I give below an extract from mine:

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

This class uses the Windows registry to persist the properties, but you can use a database or a settings file if you prefer. The advantage of using a static class in this way is that application wide changes can be made not only at design-time, but also by the user at run-time. I always include a form in my applications allowing the user to change the preferred values. The save function not only saves to the Registry (or database etc), but it also at run-time changes the propeties in the static class. Note that static properties of a static class are not constant; in this sense they may be regarded as application wide variables. This means that any form opened subsequent to the changes being saved will immediately be affected by any changes saved.

You will easily be able to think of other application wide properties that you would like to be configurable in the same way. Fonts are another very good example.

#### **NumberBox**

Often you will want to have an input box that takes numbers only. Again by deriving from the standard controls this is easily achieved, for example:

```
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 (\text{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) != 1)
            {
               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++)</pre>
                {
                    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++)</pre>
        {
            decFormat += "0";
        }
        if (extraDecPlaces > 0)
        {
            for (int i = 0; i < extraDecPlaces; i++)</pre>
            {
                decFormat += "#";
            }
        }
    }
    else if (extraDecPlaces > 0)
    {
       decFormat += ".";
       for (int i = 0; i < extraDecPlaces; i++)</pre>
        {
            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 (1 > 0)
        {
            char[] c = amountBox.ToCharArray();
            char endChar = c[1 - 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 = 100000.0;
                    else
                    {
                        string s = amountBox.Substring(0, 1 - 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, 1 - 1);
                       stripNonNumerics(ref s);
                       d = Double.Parse(s, nF) * 1000.0;
                    }
                   break;
               }
           default:
               {
                    //remove offending char
                    string s = amountBox.Substring(0, 1 - 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 (1 > 0)
        {
            char[] c = amountBox.ToCharArray();
            for (int i = 0; i < 1; 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;
   }
}
```

As well as restricting input to numbers, this class has a few special features. It exposes a property Value to represent the double value of the number, it formats the text, optionally with thousand separators, and it provides short-hand entry of large numbers: 10M expands on leave to 10,000,000.00 (the number of decimal places being a property). For the sake of brevity, the decimal and thousand separators have been hard-coded. In a production system, these are also user preferences.

Read Inheriting Controls online: https://riptutorial.com/winforms/topic/6476/inheriting-controls

## Chapter 6: Showing a form

### Introduction

This topic explains how the WinForms engine works to display forms and how you control their lifetimes.

### **Examples**

Show a modeless or a modal form

After defining the structure of your form with the WinForms designer you can display your forms in code with two different methods.

• Method - A Modeless form

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

• Method - A Modal Dialog

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

The two methods have a very important distinction. The first method (the modeless one) shows your form and then returns immediately without waiting the closure of the just opened form. So your code continues with whatever follows the Show call. The second method instead (the modal one) opens the form and blocks any activity on the whole application until you close the form via the close button or with some buttons appropriately configured to close the form

#### **Closing a modeless form**

A modeless form is employed (usually) when you need to shows something permanentely alongside your application main screen (think about a legend or an view on a stream of data coming asynchronously from a device or an MDI Child Window).

But a modeless form poses an unique challenge when you want to close it. How to retrieve the instance and call the Close method in that instance?

You can keep a global variable referencing the instance you want to close.

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

But we can also choose to use the Application.OpenForms collection where the form engine stores all the form instances created and still open.

You can retrieve that particular instance from this collection and call the Close method

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

**Closing a modal form** 

When a form is shown using the *showDialog* method, it is necessary to set the form's *DialogResult* property to close to form. This property can be set using the enum that's also called *DialogResult*.

To close a form, you just need to set the form's DialogResult property (to any value by DialogResult.None) in some event handler. When your code exits from the event handler the WinForm engine will hide the form and the code that follows the initial ShowDialog method call will continue execution.

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

The calling code can capture the return value from ShowDialog to determine what button the user clicked in the form. When displayed using showDialog(), the form is not disposed of automatically (since it was simply hidden and not closed), so it is important to use an using block to ensure the form is disposed.

Below is an example of checking the result of using the built-in <code>OpenFileDialog</code>, checking the result, and accessing a property from the dialog before disposing it.

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

You can also set the DialogResult property on a button. Clicking that button will set the DialogResult property on the form to the value associated with the button. This allows you close the form without adding an event handler to set the DialogResult in the code.

For example, if you add an OK button to your form and sets its property to DialogResult.OK then the form closes automatically when you press that button and the calling code receives a DialogResult.OK in return from the ShowDialog() method call.

Read Showing a form online: https://riptutorial.com/winforms/topic/8768/showing-a-form

## **Chapter 7: TextBox**

## Examples

Auto completion from a collection of strings

```
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);
```

This will autocomplete the as the user tries to type G or L.

AutoCompleteMode.SuggestAppend will both display a list of suggested values and it will auto type the first match, Append only and Suggest only are available, too.

#### Allow only digits in the text

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

This will only permit the use of digits and control characters in the TextBox, other combinations are possible using the same approach of setting the Handle property to true to block the text.

The user can still copy/paste unwanted characters so an additional check should be on the TextChanged to cleanse the input:

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

In this example a Regular expression is used to filter the text.

NumericUpDown should be preferred for numbers when possible.

How to scroll to the end

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

Applying the same principle, SelectionStart can be set to 0 to scroll to the top or to a specific

number to go to a specific character.

#### Adding a Placeholder to textbox

This code places the hint text at form load and manipulates it as follows:

#### 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 If
End If
End Sub
```

Read TextBox online: https://riptutorial.com/winforms/topic/4674/textbox

## Credits

S. No	Chapters	Contributors
1	Getting started with winforms	4444, Bjørn-Roger Kringsjå, Chris Shao, Cody Gray, Community, Reza Aghaei
2	Basic controls	Aimnox, Berkay, help-info.de, Jeff Bridgman
3	Databinding	Kai Thoma
4	Help Integration	help-info.de, Reza Aghaei
5	Inheriting Controls	Balagurunathan Marimuthu
6	Showing a form	Cody Gray, Jeff Bridgman, Steve
7	TextBox	gplumb, Jones Joseph, Stefano d'Antonio