Basics of Application Development Using Windows Forms

Lesson 4

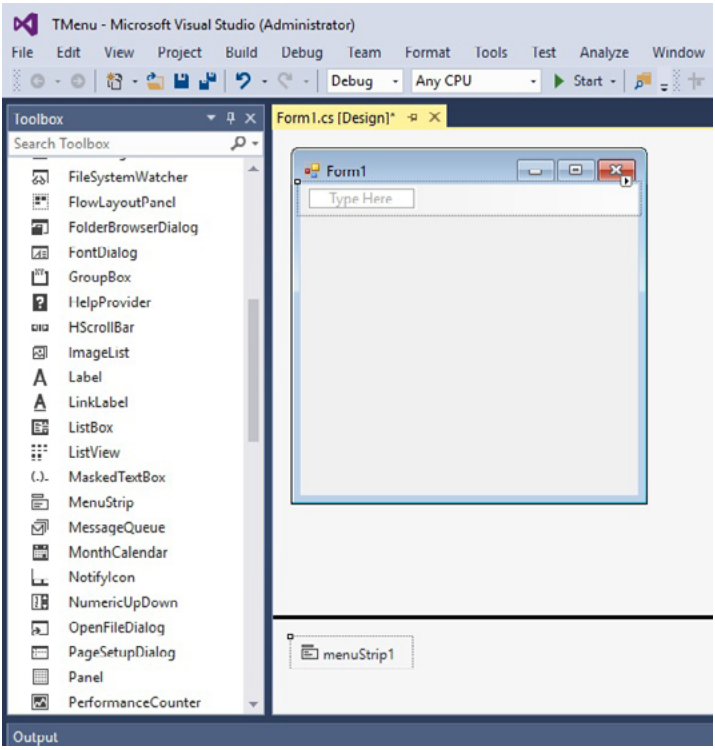
Menu

Contents

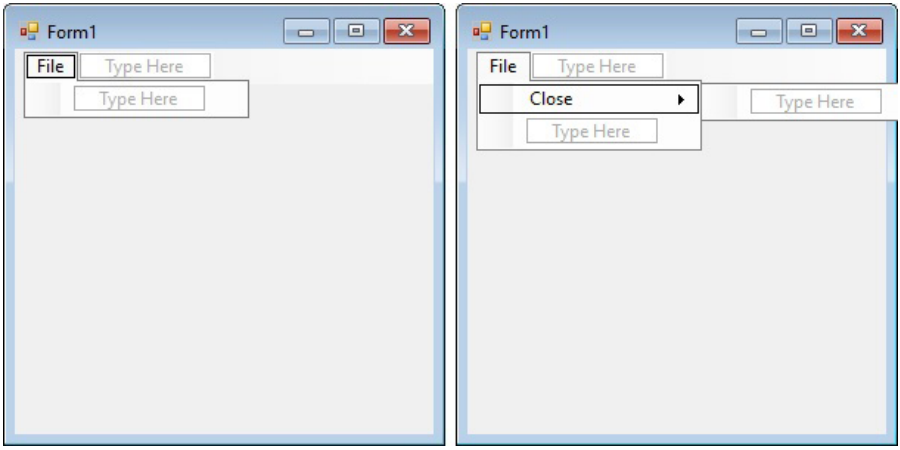
1. Creating and Using a Menu

Most of today's applications have both menus and toolbars that contain more or less the same set of commands. Menu gives the user the ability to "research" an application, to explore the available commands; and a toolbar is designed for quick access to the commands used most often.

To create a menu, you need to find a **MenuStrip** control in the **ToolBox** window and drag it to your form.



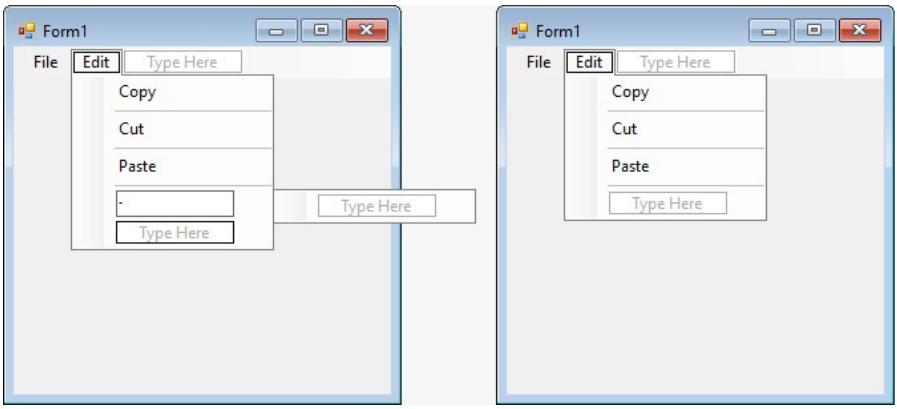
Below the form, you will see your menu. Now you can fill it. As soon as you enter a text to the first item, the menu will automatically expand by one item down and to the right.



To add a handler for a menu item, it is enough to double-click on it with the left mouse button.

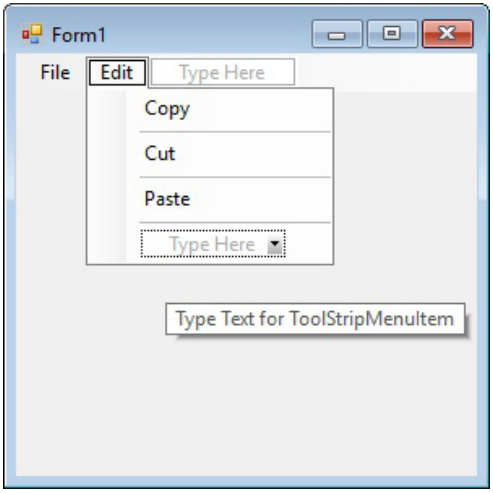
```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }

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

A menu item may be a plain text, a drop-down list, a box for entering values or a separator. By default, when you enter a text, a plain text is created. To convert a menu item into a separator, it is enough to type the character "-" when entering a text.



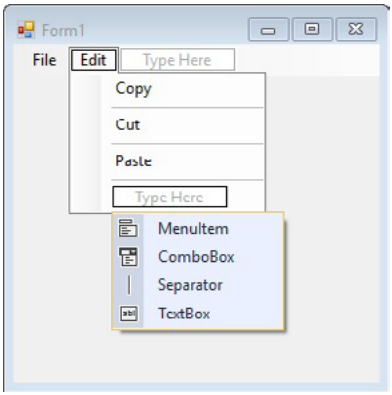
Choose one of four menu types as follows:

- Move the mouse cursor to the future menu item

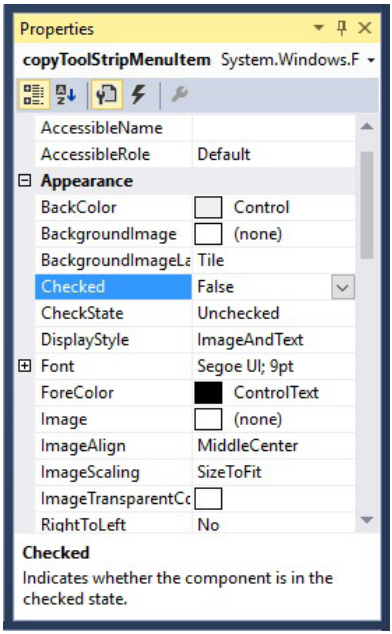


A drop-down list button will appear

- Click this button and select the desired item from the list

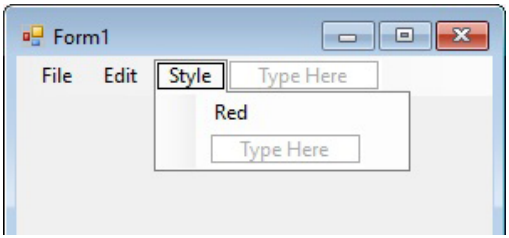


Each menu item can be marked with a tick, it is necessary to go to the **Properties** window and set the Checked property to true.



If you want to check and uncheck a check box by the click, it is necessary to set also the **CheckOnClick** property to true.

Let's look at an example. the application contains the following menu:

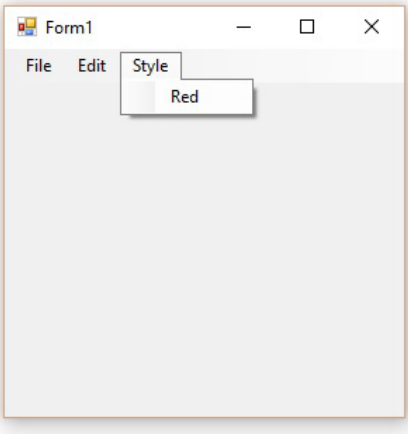


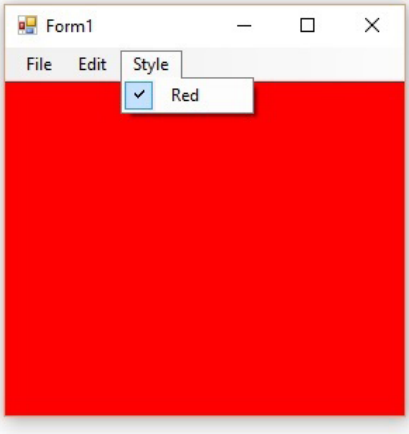
Set the **CheckOnClick** property to true for the Red menu item. Selecting this menu item, we will change the form background color to red.

```
public partial class Form1 : Form
{
    Color c;
    public Form1()
    {
        InitializeComponent();
        c = this.BackColor; //remember
                            //the current form color
}

    private void closeToolStripMenuItem_
        Click(object sender, EventArgs e)
    {
        this.Close();
    }

    private void redToolStripMenuItem_
        Click(object sender, EventArgs e)
    {
```

```
        ToolStripMenuItem it =
                        (ToolStripMenuItem) sender;
        // if checked - change the background
        //color to red
        if (it.Checked == true)
        {
            this.BackColor = Color.Red;
        }
        else //change the background color back
             //to gray
        {
            this.BackColor = c;
        }
    }
}
```

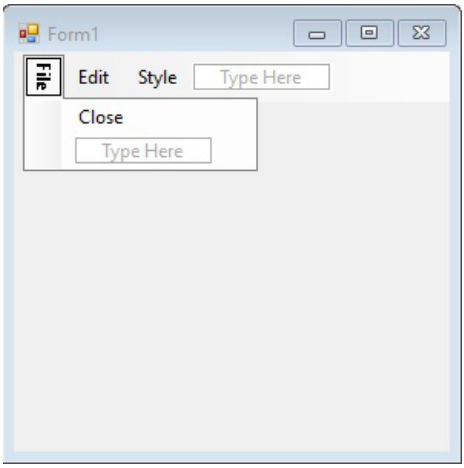
Here's the result:





When creating a menu, you can also specify a text orientation by setting the **TextDirection** property.

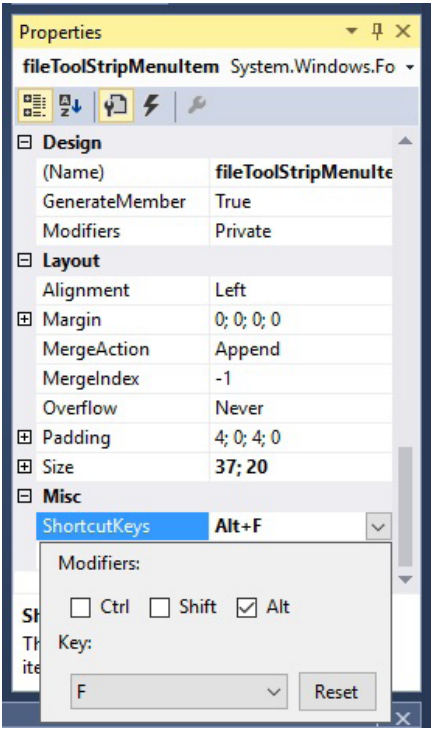
And you can set an image using the Image property.



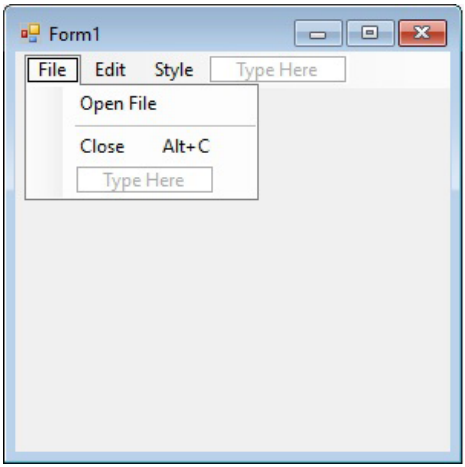
2. Accelerators

Accelerator is a certain combination of keys that duplicate a menu command. For example, you can create a **Ctrl+D** accelerator for some menu — this means that the same action will be performed both via selecting this menu item with the mouse and via pressing the key shortcut **Ctrl+D**.

To bind an accelerator to a menu item, you should call the properties window for a menu item and set the ShortcutKeys property. You can choose any key and any key combination with **Ctrl**, **Shift** or **Alt**.



In order to inform the user what accelerator is associated with a particular menu item, you can set the **ShowShortcutKeys** property to true. But brevity is important for the top level of the menu (in order to fit more items), so the text of accelerators will not be shown for it. For the top level you can make a hint, underlining the desired letter, this can be done by typing the symbol "&" before the desired letter. This method tells the user that this menu item can also be called using the combination of **Alt** and the underlined key.



3. MenuStrip Class

The **MenuStrip** class is responsible for the creation of the menu. Let's look at it closer.

The **MenuStrip** control is a container the form menu structure. **MenuStrip** can contain the following items:

- **ToolStripMenuItem**
- **ToolStripTextBox**
- **ToolStripComboBox**

You can add the **ToolStripMenuItem** objects to the **MenuStrip** object that represents individual commands in the menu structure. Each **ToolStripMenuItem** object can be a command for an application or a parent menu for other elements of the nested menu.

Basic methods of the **MenuStrip** class are as follows:

Method	Description
`Contains`	Returns true if the menu contains the specified control.
`GetItemAt`	Returns the item located at the specified point in the client area.
`Hide`	Conceals the control.
`Refresh`	Forces to redraw itself and all the child controls

Basic properties of the **MenuStrip** class are as follows:

Properties	Description
`Items`	Gets all the items that belong to a ToolStrip object.
`MdiWindowsListItem`	Gets or sets the ToolStripMenuItem that is used to display a list of MDI child forms.
`Name`	Gets or sets the name of the object.
`Parent`	Gets or sets the parent container of the control.
`Visible`	Gets or sets a value indicating whether the control and all its child controls are displayed.

Basic properties of the **ToolStripMenuItem** class are as follows:

Method	Description
`Checked`	Gets or sets a value indicating whether the ToolStripMenuItem is checked.
`CheckOnClick`	Gets or sets a value indicating whether the ToolStripMenuItem should automatically appear checked and unchecked when clicked.
`CheckState`	Gets or sets a value indicating whether a ToolStripMenuItem is in the checked, unchecked, or indeterminate state.
`DisplayStyle`	Gets or sets whether text and images are displayed on a ToolStripItem.
`DropDownItems`	Gets the collection of items in the ToolStripDropDown that is associated with this item.

Method	Description
`Name`	Gets or sets the name of the item.
`Parent`	Gets or sets the parent container of the control.
`Visible`	Gets or sets a value indicating whether the control and all its child controls are displayed.

4. Creating a Template-Based Menu

Quite often, in the program it is necessary to implement a menu in different languages or to change a menu depending on the user rights or input data. One of the ways to implement this functionality is to create several kinds of menus, and to arrange a "substitution" of the menu, if needed.

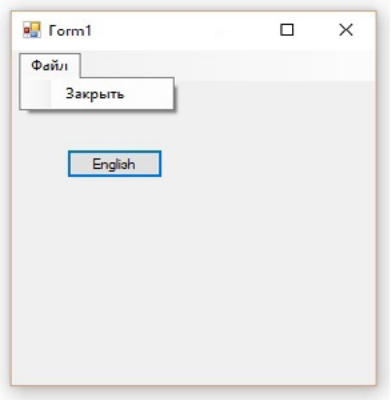
Let's consider an example. Create a form and two types of menus — English and Russian.

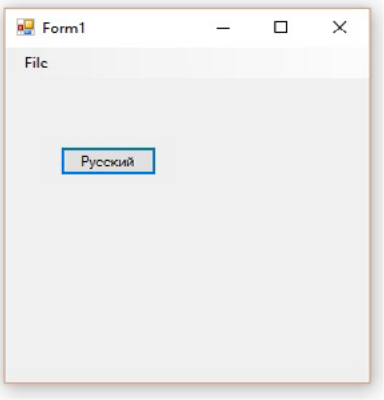
Add a button, which will implement the menu "substitution", to the form.

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        menuStripEnglish.Visible = false;
        button1.Text = "English";
    }

    private void closeToolStripMenuItem_Click(object
        sender, EventArgs e)
   {
```

```
        this.Close();
    }

    private void button1_Click(object sender,
        EventArgs e)
    {
        if (button1.Text.CompareTo("English") == 0)
        {
            button1.Text = "Russian";
            menuStripEnglish.Visible = true;
            menuStripRussian.Visible = false;
            this.MainMenuStrip = menuStripEnglish;
        }
        else
        {
            button1.Text = "English";
            menuStripEnglish.Visible = false;
            menuStripRussian.Visible = true;
            this.MainMenuStrip = menuStripRussian;
        }
    }
}
```

Here's the result:





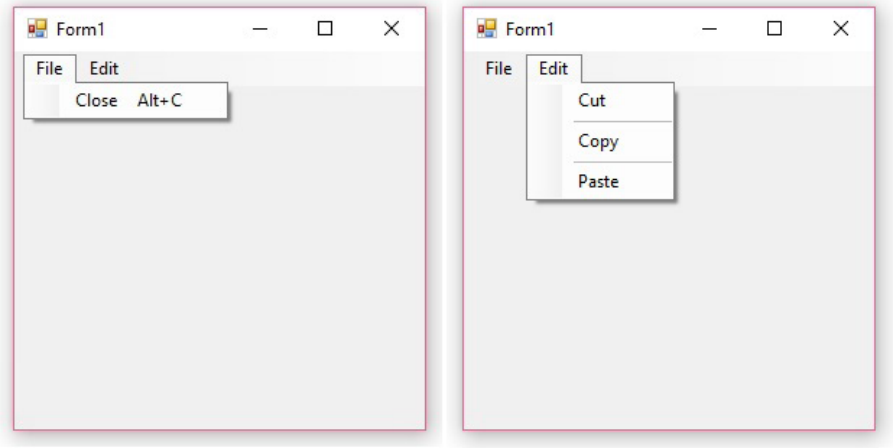
5. Dynamic Creation of a Menu

Any menu can be created dynamically. Let's consider an example of creating a simple menu dynamically:

```
public partial class Form1 : Form
{
    MenuStrip m;
    public Form1()
    {
        InitializeComponent();
        m = new MenuStrip();
        //add a top-level menu
        ToolStripMenuItem file= (ToolStripMenuItem)
            m.Items.Add("File");
        ToolStripMenuItem edit = (ToolStripMenuItem)
            m.Items.Add("Edit");
        this.MainMenuStrip = m;
        this.Controls.Add(m); //add a menu to the form

        //add a drop-down menu for the Edit item
        edit.DropDownItems.Add("Cut");
        //add a separator
        edit.DropDownItems.Add(new ToolStripSeparator());
        edit.DropDownItems.Add("Copy");
        //add a separator
        edit.DropDownItems.Add(new ToolStripSeparator());
        edit.DropDownItems.Add("Pastle");
        // add a drop-down menu for the File item
        ToolStripMenuItem close = (ToolStripMenuItem)
            file.DropDownItems.Add("Close");
        //connect the menu with the Alt+C accelerator
        close.ShortcutKeys = Keys.Alt | Keys.C;
```

```
        close.ShowShortcutKeys = true; //display
                                       //accelerators
        //add a handler to the Close menu item
        close.Click += new EventHandler(close_Click);
    }

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

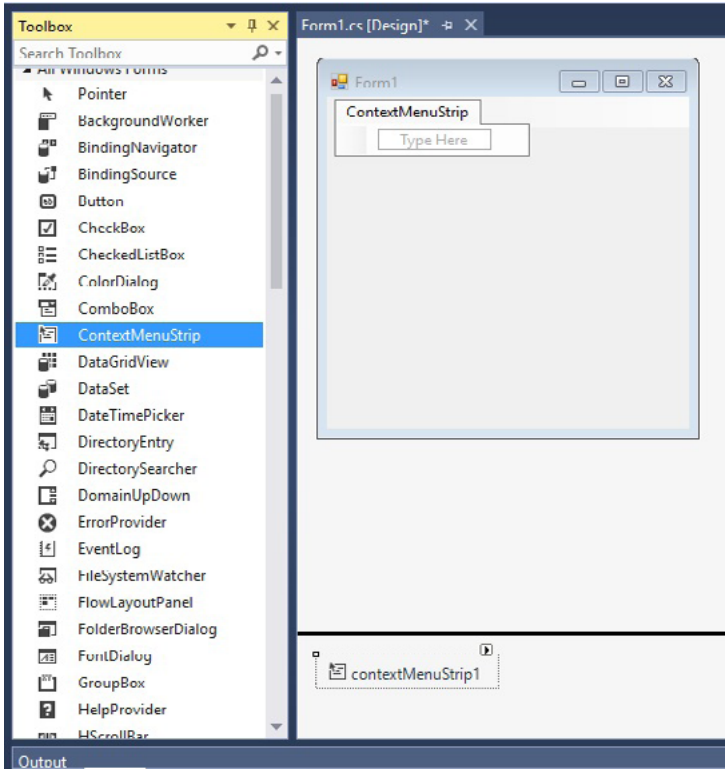
Here's the result:



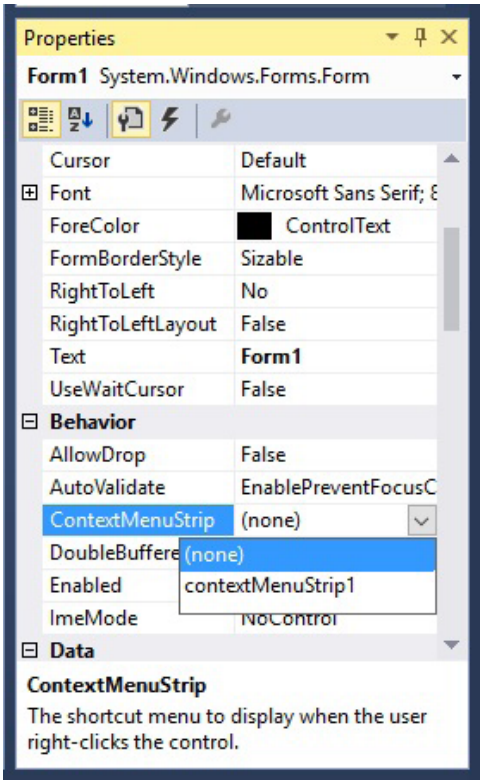
6. Context Menu. ContextMenuStrip Class

Context menu is a menu that appears when clicking with the right mouse button. the context menu is not bound to the form but to any control, and may be different for different elements of the same form.

To add a context menu, you should drag the **ContextMenuStrip** control to the form from the **ToolBox**. the context menu appears at the bottom, under the form.



The context menu cannot contain top-level elements, but the drop-down list only. Context menu are automatically tied to the form, it is necessary to set the **ContextMenuStrip** form property. the same property can be set for any control.



Basic methods of the **ContextMenuStrip** class are as follows:

Method	Description
`Contains`	Returns true if the menu contains the specified control.
`GetItemAt`	Returns the item located at the specified point in the client area.

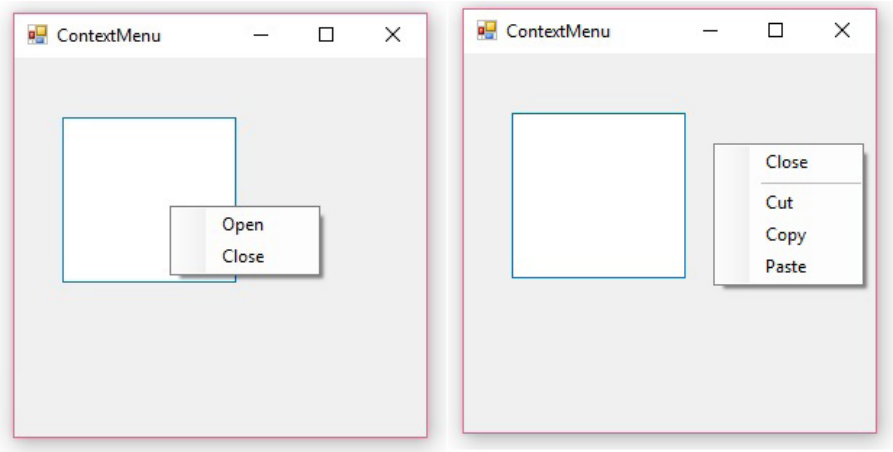
Method	Description
Hide	Conceals the control.
Refresh	Forces to redraw itself and all the child controls

Basic properties of the **ContextMenuStrip** class are as follows:

Properties	Description
Items	Gets all the items that belong to a ToolStrip object.
Name	Gets or sets the name of the object.
Parent	Gets or sets the parent container of the control.
Visible	Gets or sets a value indicating whether the control and all its child controls are displayed.

The context menu can also be created dynamically. Let's Consider an example in which a context menu is added to the **TextBox** control.

```
{
ContextMenuStrip m;
    public Form1()
    {
        InitializeComponent();
        m = new ContextMenuStrip();
        m.Items.Add("Open");
        m.Items.Add("Close");
        textBox1.ContextMenuStrip = m;
    }
}
```


ContextMenu
Open
Close
ContextMenu
Close
Cut
Copy
Paste

7. Toolbar. ToolStrip Class

ToolBar is a **Windows** toolbar. Although a **ToolStrip** control replaces the **ToolBar** control of the previous versions and adds functionality if necessary, the **ToolBar** control is retained for both backward compatibility and future use.

ToolBar controls are used to display **ToolBarButton** controls that may be represented by usual buttons, radio-buttons or buttons with drop-down list. You can assign an image for buttons. To do this, create an **ImageList** object, assign it to the **ImageList** property of the toolbar, and then assign the image index value to the **ImageIndex** property of each **ToolBarButton**. Then you can specify a text that will be displayed below the image or to the right of it y setting the Text property of the **ToolBarButton** object. Toolbar buttons can be divided into logical groups by separators. Separator is a toolbar button that has the **Style** property with the **ToolBarButtonStyle.Separator** value. Only one handler can be set for all **ToolBar** elements.

Let's consider an example that implements a simple text editor. For this, place a **TextBox** on the form, set the **MultiLine** property to **true** and **Dock** property to **Fill** in order it fills the whole form. **ToolBox** will be created dynamically consisting of three buttons: **Open**, **Save**, **Exit**.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
```

```
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace CreateToolBar
{
    public partial class Form1 : Form
    {
        ToolBar tBar;
        ImageList list;
        public Form1()
    {
        InitializeComponent();
        list = new ImageList();
        list.ImageSize = new Size(50, 50);
        list.Images.Add(new Bitmap("open.bmp"));
        list.Images.Add(new Bitmap("save.bmp"));
        list.Images.Add(new Bitmap("exit.bmp"));

        tBar = new ToolBar();

        tBar.ImageList = list; //bind a list of
                               //images to the toolbar
        ToolBarButton toolBarButton1 =
                                 new ToolBarButton();
        ToolBarButton toolBarButton2 =
                                 new ToolBarButton();
        ToolBarButton toolBarButton3 =
                                 new ToolBarButton();
        ToolBarButton separator =
                                 new ToolBarButton();
        separator.Style = ToolBarButtonStyle.
                                 Separator;

        toolBarButton1.ImageIndex = 0; //Open
        toolBarButton2.ImageIndex = 1;// save
        toolBarButton3.ImageIndex = 2; //exit
```

```
        tBar.Buttons.Add(toolBarButton1);
        tBar.Buttons.Add(separator);
        tBar.Buttons.Add(toolBarButton2);
        tBar.Buttons.Add(separator);
        tBar.Buttons.Add(toolBarButton3);

        tBar.Appearance = ToolBarAppearance.Flat;
        tBar.BorderStyle = BorderStyle.Fixed3D;
        tBar.ButtonClick += new
            ToolBarButtonClickEventHandler(tBar_
            ButtonClick);

        this.Controls.Add(tBar);

    }

    void tBar_ButtonClick(object sender,
        ToolBarButtonClickEventArgs e)
    {
        OpenFileDialog f1;
        SaveFileDialog f2;
        switch (e.Button.ImageIndex)
        {
            case 0:

                f1 = new OpenFileDialog();
                if (f1.ShowDialog() == DialogResult.OK)
                {
                    StreamReader r=
                        File.OpenText(f1.FileName);
                    textBox1.Text = r.ReadToEnd();
                }
                break;
            case 1:
                f2 = new SaveFileDialog();
                if (f2.ShowDialog() == DialogResult.OK)
                {
```

```
                    StreamWriter w =
                        new StreamWriter(f2.FileName);
                    w.WriteLine(textBox1.Text);
                    w.Close();
                }
                break;
            case 2:
                Close();
                break;
            }
        }
    }
}
```

ToolStrip is a base class for MenuStrip StatusStrip and ContextMenuStrip classes. the **ToolStrip** class provides a set of components that provide the control of drawing, input via keyboard and mouse, as well as drag and drop. ToolStrip can contain one of the following controls:

- **ToolStripButton**

- **ToolStripSeparator**
- **ToolStripLabel**
- **ToolStripDropDownButton**
- **ToolStripSplitButton**
- **ToolStripTextBox**
- **ToolStripComboBox**

8. Home Task

I. Develop a text editor; arrange opening \ saving text files.

- Toolbar should contain the following buttons: open, save new document, copy, cut, paste, undo, editor settings (font color, background color, font).
- Menu should duplicate the toolbar (+ select all, + Save as).
- In the window title there should be a full path to the file.
- Arrange a context menu for the editor window (Copy, Cut, Paste, Cancel).

II. Write a program "Explorer"

- At the first start, the program displays a list of available disks.
- The program should contain a disk tree, address bar, menu, toolbar and a window to display the folder contents.
- Disk Tree displays only the drives and folders (can be implemented using the ListBox).
- When you double-click on the folder — files and subfolders are displayed in the content window.
- The program should have a detailed menu, a context menu and the ability to work with hotkeys.