Windows Forms
(Microsoft Visual Studio 2019) - Tutorial

March 2021

Creating Windows Forms Applications

1. Create a new project

   (a) Choose **Windows Forms App** (WITHOUT .NET Framework in title). Project template can be found with filters:

   - Language → C#
   - Platform → Windows
   - Project type → Desktop

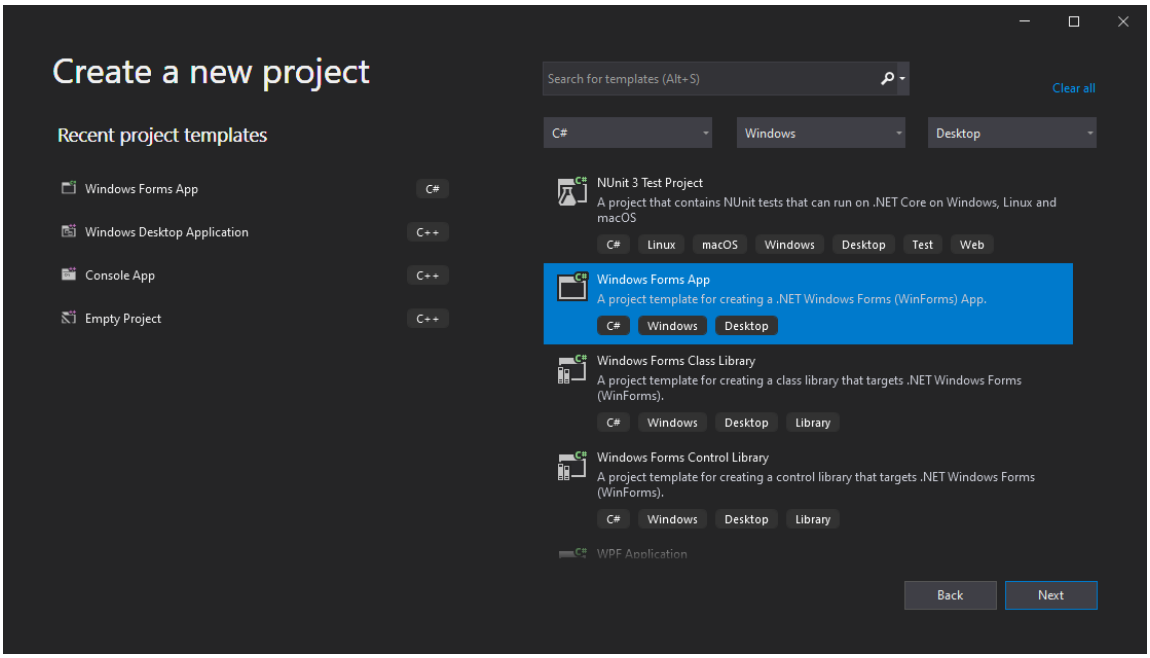


Figure 1: VS 2019 - Choosing project template

(b) Choose name and location for your project

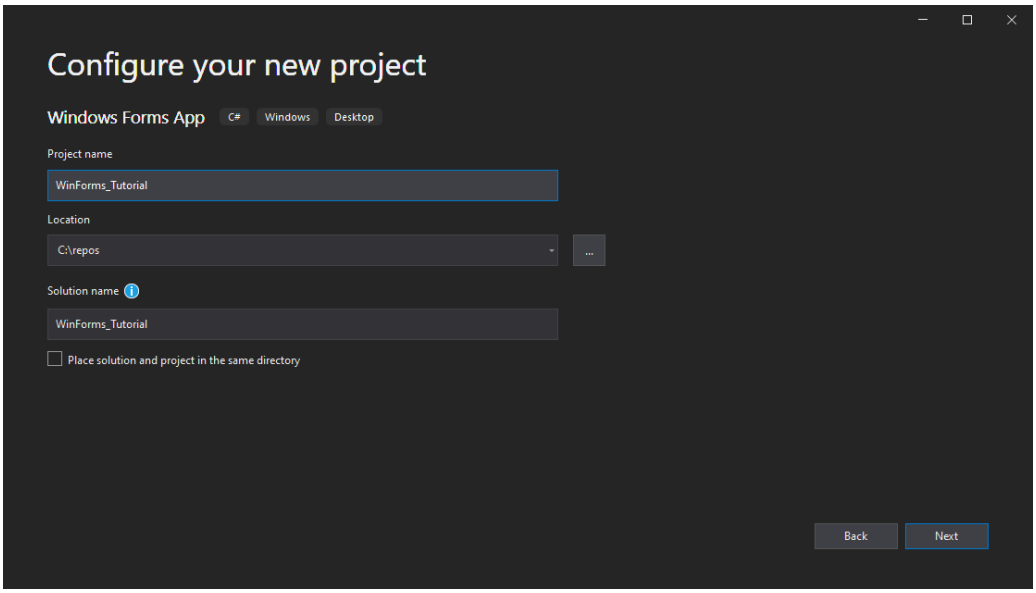


Figure 2: VS 2019 - Choosing project name

(c) Switch Target Framework to **.NET 5.0**

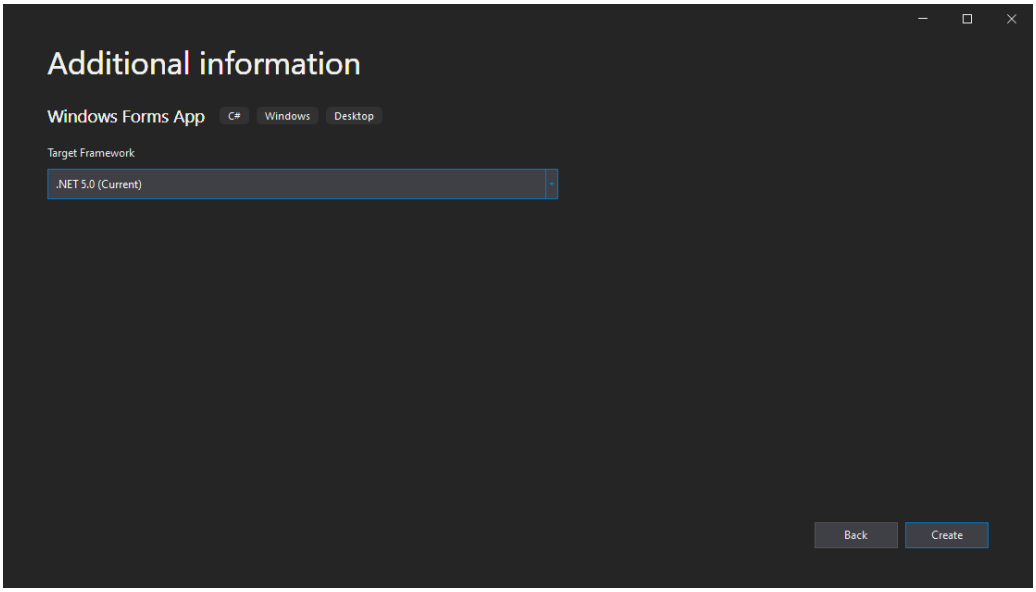


Figure 3: VS 2019 - Choosing Target Framework

2. Compile and get familiar with generated files

   (a) Compile and run application. Note that one empty window will appear. Close it and check the generated files.

   (b) Investigate file **"Program.cs"**. It contains the **"Main"** method, the entry point for your application.

```
1  static void Main()
2  {
3      Application.SetHighDpiMode(HighDpiMode.SystemAware);
4      Application.EnableVisualStyles();
5      Application.SetCompatibleTextRenderingDefault(false);
6      Application.Run(new Form1());
7  }
```

   The most important line is `Application.Run(new Form1())`, which starts application message loop and shows Form1 form.

   (c) Open file **"Form1.cs"**. Note that instead of code you will see Form Designer. It allows you to add, remove, edit controls on your forms using graphics interface.

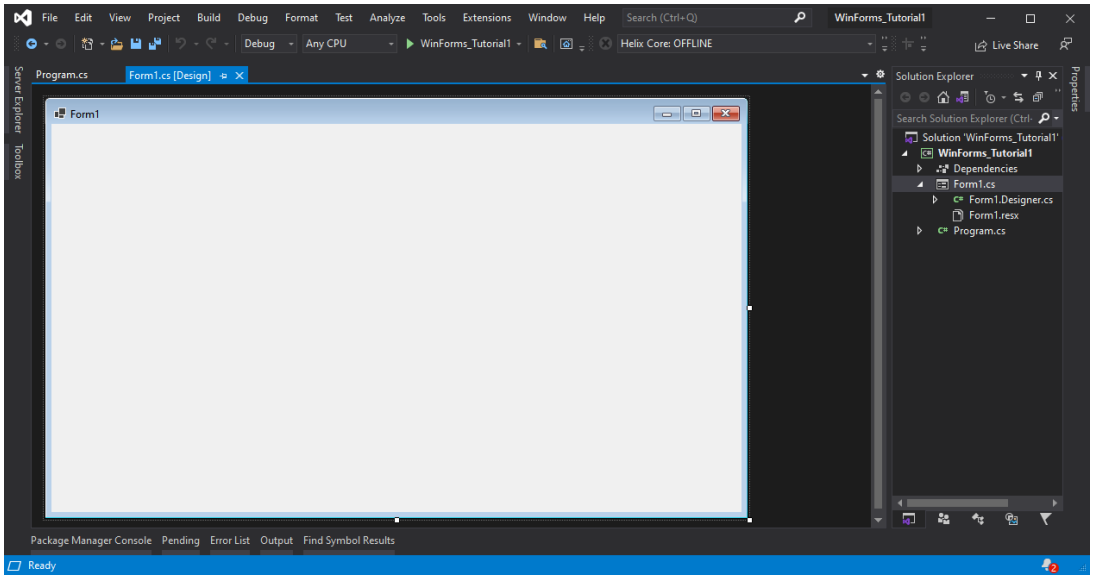


Figure 4: VS 2019 - Window designer

   (d) Right-click **"Form1.cs"** in solution explorer and click **View Code**. You will see "Code-behind" of the form:

```
1  public partial class Form1 : Form
2  {
3      public Form1()
4      {
5          InitializeComponent();
6      }
7  }
```

It's a place where you should write logic for the form. Note, that Form1 class is derived from the **Form class** – one of the fundamental classes of Windows Forms. Note also that Form1 class is marked **partial**.

(e) Open **"Form1.Designer.cs"** – under **"Form1.cs"** tree. This file is used by Form Designer. It's second part of Form1 class. You should't change this file, unless you know what you are doing. All changes to this file will be lost after using Form Designer – so don't place your code here!

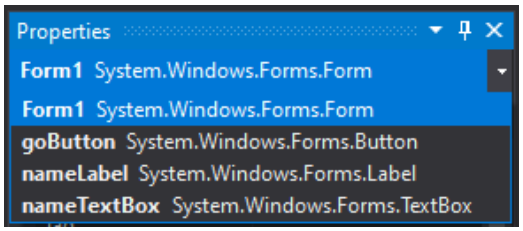
3. Add controls to the form

(a) Add controls to Form. Open Form Designer. Open Toolbox (**View → Toolbox**). You can see there all available controls. Find and place on the Form the following controls: **Label**, **Textbox**, **Button**.

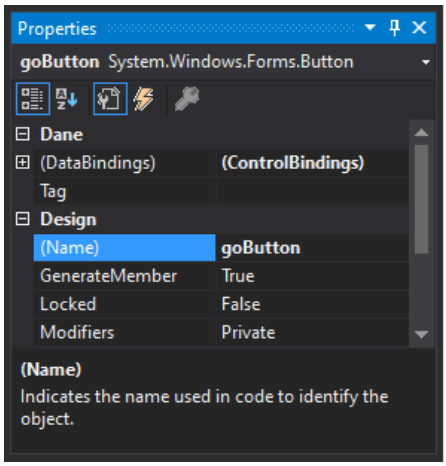
(b) Show control properties. Right click on **Label** control and choose **Properties**. In properties box, you can adjust controls. Go through the list and make yourself familiar with available options. Remember that this list may be different for different controls.

(c) Change names of controls. The **Name** is common for all controls and it's a name of control to use in "Code-Behind". Name of each control on the form must be unique.

- Set **Name** of **Label** to **"nameLabel"**
- Set **Name** of **Textbox** to **"nameTextBox"**
- Set **Name** of **Button** to **"goButton"**



(a) List of all controls in the window



(b) Properties of the button

Figure 5: Control's properties

(d) Set other properties:

- Change size of **Form** to 200x200. Use **Size** property or mouse in Form Designer

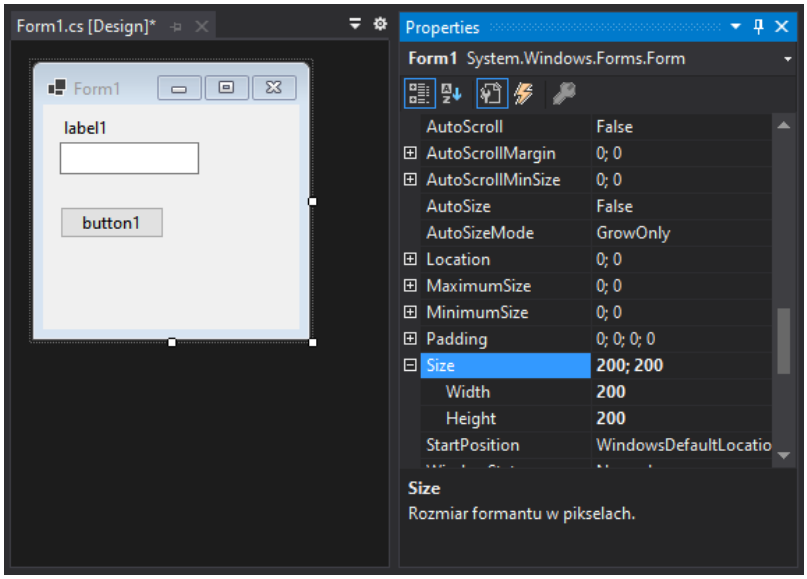


Figure 6: Form Size property

- Set **Text** of **Form** to **"Main Form"**
- Set **Text** of **Button** to **"OK"**
- Set **Text** of **Label** to **"Your name:"**

Note: Control's text is automatically updated.

(e) Run your application. You will see a window with control, however, pressing a button does nothing. Close your application.

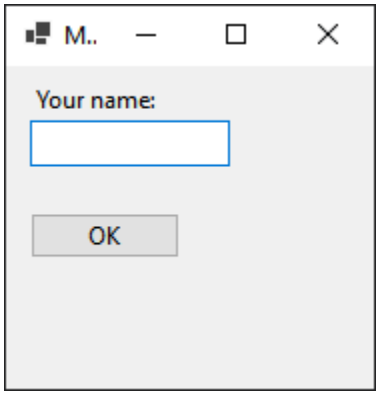


Figure 7: MainForm window

(f) Add code-behind. To respond to user's interaction you must add event handlers to your controls. To see all available events for a control, select it in the Form Designer, and click **Events** button in the properties box.

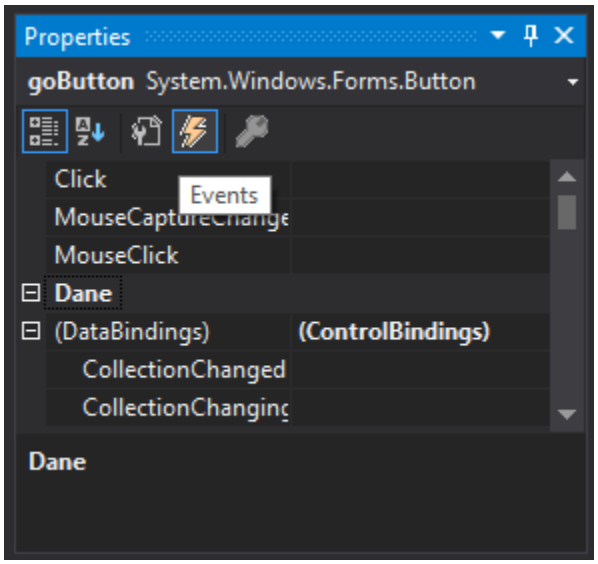


Figure 8: Button events

Note: To see properties again, press **Properties** button.

To create an event handler for button's Click event, double click in **Click** event box. Your code-behind file opens and you will see a new empty method generated for you.

```
1 private void goButton_Click(object sender, EventArgs e)
2 {
3 }
```

Note: Each control has its **"default"** event. When you double click it in **Form Designer** it will automatically create a handler for this event. For Button its **Click** event. So instead of opening Events box you could just double click button in Form Designer.

Put this line in the handler:

```
1 MessageBox.Show("Welcome " + nameTextBox.Text);
```

`MessageBox.Show` creates a message box with "OK" button. Note that we use nameTextBox.Text to get **Textbox** input text

(g) Run your application, write your name in textbox and press OK.

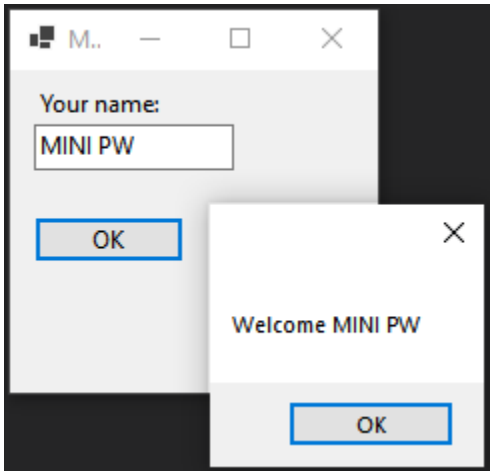


Figure 9: Main form and message box

(h) Adding error notification and Validation.

- Add **Error Provider** control to your form
- Name it **"nameErrorProvider"**. **Note:** this component isn't placed on your form but it's located bellow.

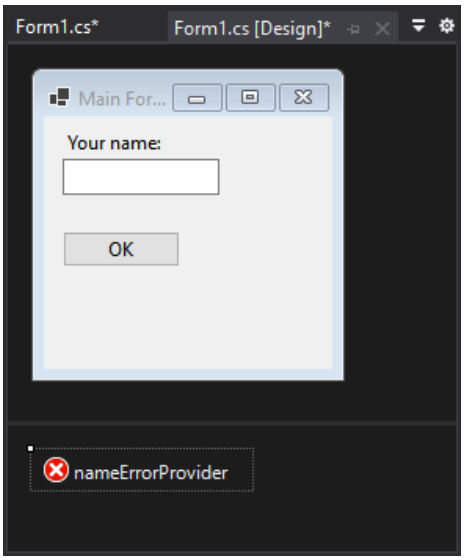


Figure 10: Error provider

(i) Add **Validation** event handler for nameTextBox (**Events → Focus → Validating**):

```
private void nameTextBox_Validating(object sender, CancelEventArgs e)
{
    if (nameTextBox.Text.Length == 0 || nameTextBox.Text.Length > 6)
    {
        nameErrorProvider.SetError(nameTextBox,
            "Name must contain from 1 to 6 characters");
        e.Cancel = true;
        return;
    }
    nameErrorProvider.SetError(nameTextBox, string.Empty);
    e.Cancel = false;
}
```

(j) Change your **goButton_Click** method to the following:

```
private void goButton_Click(object sender, EventArgs e)
{
    if (this.ValidateChildren())
        MessageBox.Show("Welcome " + nameTextBox.Text);
}
```

(k) Run your application. Leave empty textbox and click "OK". Note that the message box doesn't show up. Instead you can see error icon with error message next to textbox. Write something correct in textbox and see that icon hides and message box shows up.

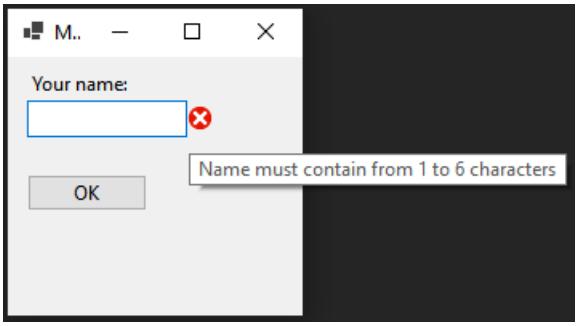


Figure 11: Textbox error icon

(l) Disable **AutoValidation**. You may find impossible to close window when textbox is invalid. To correct that set Form1 property **AutoValidate** to **Disable** (**Properties** → **Behavior** → **AutoValidate**).

Using Controls – Calendar Manager

1. Create new project, name it "WinFormsCalendar"
2. Create layout
   - Set **Form** defaults
     - Set **Size** of the form to **800 x 500**
     - Set **StartPosition** to **Center Screen**
   - Add a **Split Container** to the form. Change its orientation to horizontal using menu in the top-right corner of the control. Rename it to "splitContainer".

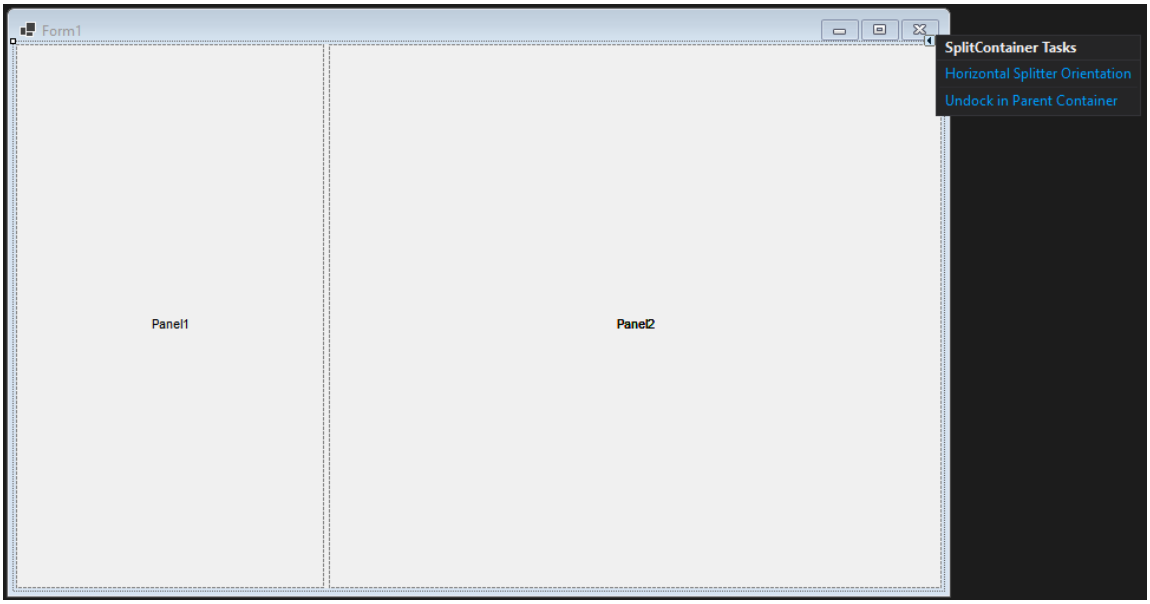


Figure 12: Split container Tasks

   - Change color of the **Split Container**. Split container contains 2 panels and a splitter between them. It allows user to change size of the panels by moving the splitter. However, splitter's color is the same as panels, so it is invisible. To change the color, do the following:
     - Set splitContainer **Back Color** to **LightSteelBlue**. If you have problem selecting SplitContainer in the Designer, use menu at the top of properties box.

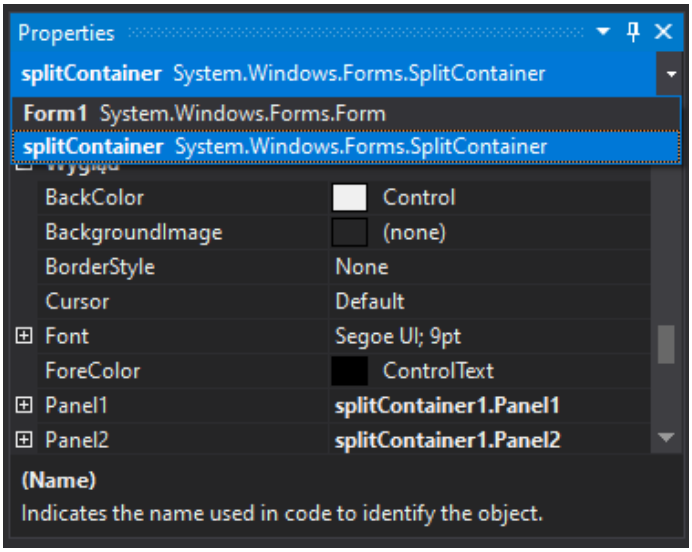


Figure 13: Split container on control list and its properties

- Color of panels also changed. Now select first panel and change its back color to **ControlLight**, the same for the second panel.
- Run the application. You should see the splitter and be able to change its position.

3. Anchoring controls

- Add two controls to the top panel: **MonthCalendar** and **TextBox**. Name them "monthCalendar" and "noteTextBox", respectively
- Set Calendar properties
  - Set **CalendarDimensions Width** to **2** (**Appearance** → **CalendarDimensions**)
  - Set **MaxSelectionCount** to **1** (**Behavior** → **MaxSelectionCount**)
- Set TextBox properties
  - Set **Multiline** to **True** (**Behavior** → **Multiline**)
- Set controls layout. Place Calendar in the top-left corner of the Panel. Resize the **Textbox** to fill the rest of it.

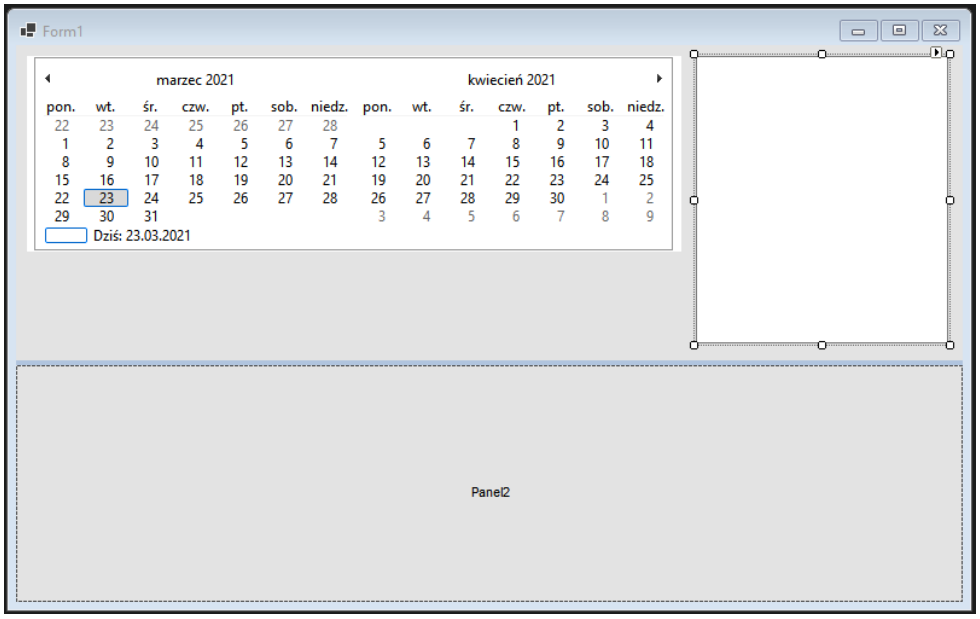


Figure 14: Calendar window layout

- Start application. Try resizing window. Notice that the size of the textbox doesn't change.
- Go back to the **Form Designer**. Change noteTextBox **Anchor** property to **Top Bottom Left Right**.
- Start Application again and see that resizing works properly.

4. Add collection control

- Bottom panel controls
  - Add 2 **buttons** and **ListView** to the bottom panel.
  - Name them **"addButton"**, **"removeButton"** and **"calendarListView"**, respectively.
  - Place them as in the screenshot bellow.

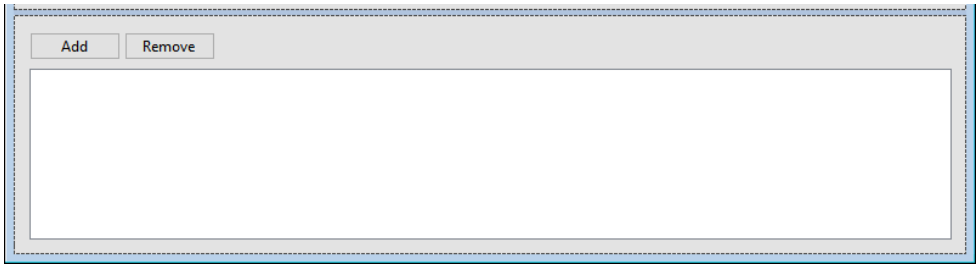


Figure 15: Window's bottom panel layout

- Set following properties
  (a) addButton **Text** to **"Add"**
  (b) removeButton **Text** to **"Remove"**
  (c) removeButton **Enabled** to **False** (**Behavior** → **Enabled**)
  (d) calendarListView **View** to **Details** (**Appearance** → **View**)
  (e) calendarListView **FullRowSelect** to **True** (**Appearance** → **FullRowSelect**)
  (f) calendarListView **Anchor** to **Top Bottom Left Right**
- Start application again and see that resizing works properly. Select calendarListView. In properties box find **Columns** and click [...] button. You will see List View column editor. Add 2 columns:
  (a) **Name**: **"dateColumnHeader"**; **Text**: **"Date"**; **Width**: **120**
  (b) **Name**: **"noteColumnHeader"**; **Text**: **"Note"**; **Width**: **200**

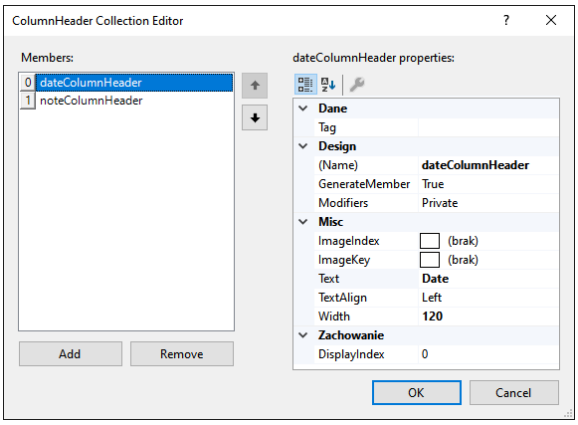


Figure 16: ListView columns' definitions

- Compile and make sure everything looks fine.

5. Add Logic to the application

- Adding item to **ListView**
  – Add **Click** event handler to addButton.
  – Fill it with following code:

```
1  private void addButton_Click(object sender, EventArgs e)
2  {
3      string date = monthCalendar.SelectionStart.ToShortDateString();
4      string note = noteTextBox.Text;
5      ListViewItem newItem = new ListViewItem(new[] { date, note });
6      calendarListView.Items.Add(newItem);
7      noteTextBox.Text = "";
8  }
```

Note that to add item to the **ListView** you must create a **ListViewItem** object. Because we have 2 columns in our **ListView**, the **ListViewItem** should have 2 values – date and note.

- Run application and try adding some items to our **ListView**.

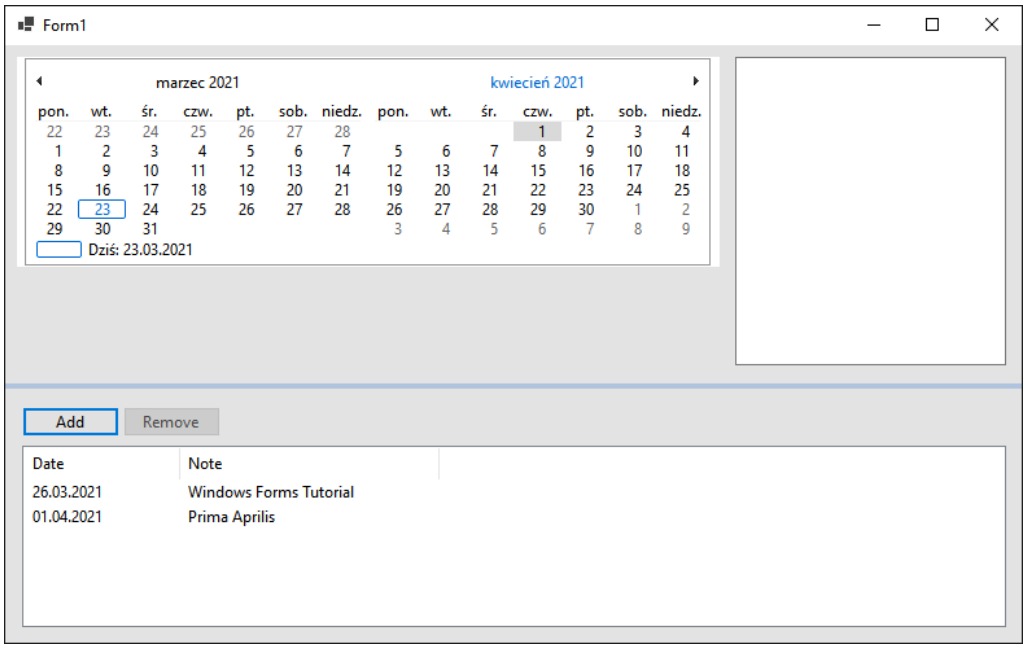


Figure 17: Calendar window

- Remove selected item from **ListView**.

```
1  private void removeButton_Click(object sender, EventArgs e)
2  {
3      foreach (ListViewItem item in calendarListView.SelectedItems)
4      calendarListView.Items.Remove(item);
5  }
```

Remember that your removeButton is disabled. Add **SelectedIndexChanged** event handler to a calendarListView:

```
1  private void calendarListView_SelectedIndexChanged(object sender, EventArgs e)
2  {
3      removeButton.Enabled = calendarListView.SelectedItems.Count > 0;
4  }
```

- Start application and test adding and removing items.

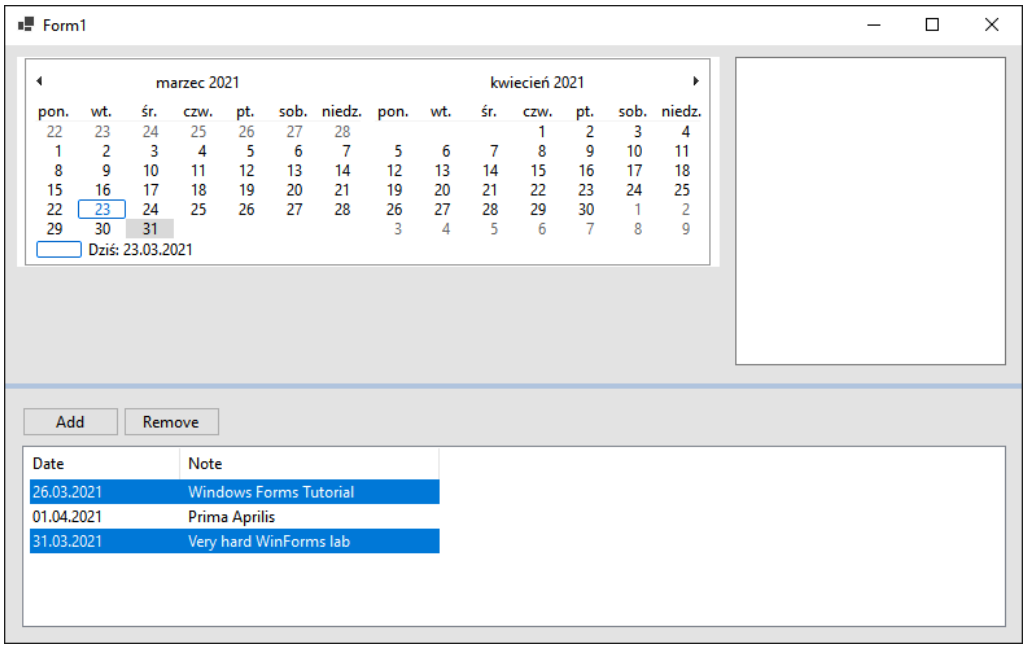


Figure 18: Removing items from Calendar

6. Adding notify Icon

   - Add **NotifyIcon** control named **"notifyIcon"** to the **Form**
     - Set **Text** to **"Windows Forms Calendar"**
     - Set **Icon** to any icon you can find on your computer
     - Set **Visible** to **True**
   - Show a **BallonTip**. Modify your **addButton_Click** method. Add at the end of it:

```
1 notifyIcon.ShowBalloonTip(1000, "Item added", "Added new event", ToolTipIcon.Info);
```

   - Run application. You will see an icon on the taskbar and a notification after adding an item.

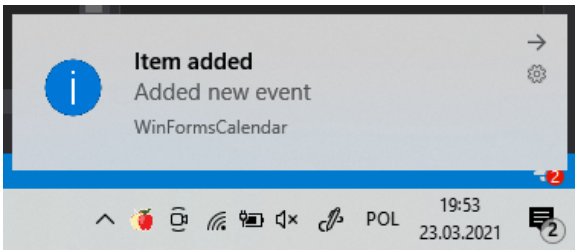


Figure 19: Calendar notification

- **Context menu:**
  - Add **ContextMenuStrip** control to our form, named **"notifyIconContextMenu"**
  - Add **Load** Event Handler to Form1:

```
private void Form1_Load(object sender, EventArgs e)
{
    notifyIconContextMenu.Items.Add("Exit", null, ExitContextMenu_Click);
}
```

  - Add Event handler for above menu item:

```
private void ExitContextMenu_Click(object sender, EventArgs e)
{
    Close();
}
```

  - Set notifyIcon's **ContextMenuStrip** property to **notifyIconContextMenu**
- Run application and right click on the notify icon.

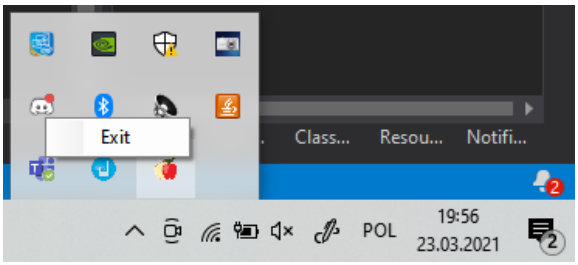


Figure 20: Menu icon Context Menu

Using Controls – WinFormsGallery

1. Create new project, name it "WinFormsGallery"
2. Create layout
   - Set Form defaults
     - Set **Size** of the form to **800 x 500**
     - Set **StartPosition** to **Center Screen**
     - Set **Locked** to **True**
   - Table Layout Panel:
     - Add a **Table Layout Panel** to the form.
     - Rename it to **"galleryLayoutPanel"**.
     - Set **Anchor** to **Top, Bottom, Left, Right**
     - Change second column **Width** to **25%**

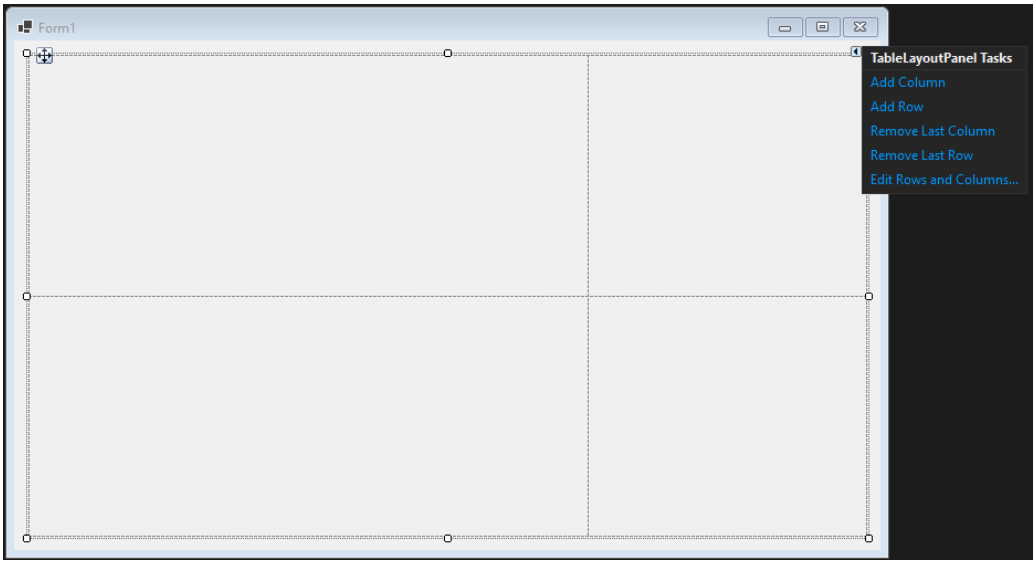


Figure 21: TableLayoutPanel columns and rows settings

- Run application and notice that nothing is visible. Table Layout Panel is just a container. We need to fill it with controls.

3. Add Buttons

   - Add Choose Image **button** to Top Right Panel
     - Change its **Name** to **"imageButton"**
     - Change its **Text** to **"Choose Image"**
     - Change its **Dock** to **Fill**
   - Add Choose Color **button** to Bottom Right Panel
     - Change its **Name** to **"colorButton"**
     - Change its **Text** to **"Choose Color"**
     - Change its **Dock** to **Fill**

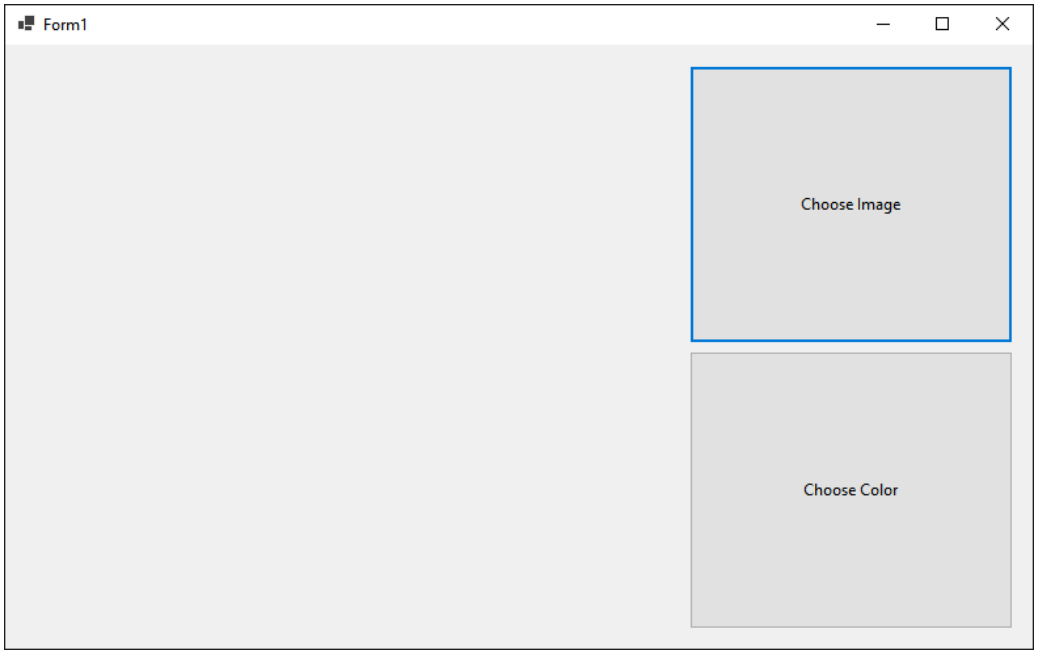


Figure 22: Application layout

4. Add Color Button logic

   - Add **Click** event to colorButton. We will change **BackColor** for bottom panels.
     - We will use **ColorDialog** to choose our color.
     - Add following code to **colorButton_Click**:

```
ColorDialog colorDialog = new ColorDialog();
if (colorDialog.ShowDialog() == DialogResult.OK)
{
    galleryLayoutPanel.GetControlFromPosition(1, 1).BackColor = colorDialog.Color;
    galleryLayoutPanel.GetControlFromPosition(0, 1).BackColor = colorDialog.Color;
}
```

   - Run application and choose your color. You will see that `System.NullReferenceException` is thrown. It is because **TableLayoutPanel** is just a container, and there is nothing inside Bottom Left panel.
   - Add Empty **Label** to Bottom Left panel. Fill it to the container. Rerun application.

Figure 23: ColorButton logic

5. Add Image Button logic

   - Add Empty **Label** to Top Left panel. Fill it to the container.
   - We will use **OpenFileDialog** to fill Top Left label background image.
     - Create **imageButton_Click** event and add following code:

```
OpenFileDialog fileDialog = new OpenFileDialog();
fileDialog.Filter = "Image Files(*.BMP;*.JPG;*.GIF)|*.BMP;*.JPG;*.GIF";
if (fileDialog.ShowDialog() == DialogResult.OK)
{
    galleryLayoutPanel.GetControlFromPosition(0, 0).BackgroundImage =
    new Bitmap(fileDialog.FileName);
    galleryLayoutPanel.GetControlFromPosition(0, 0).BackgroundImageLayout =
    ImageLayout.Stretch;
}
```

   - Run application and choose an image.

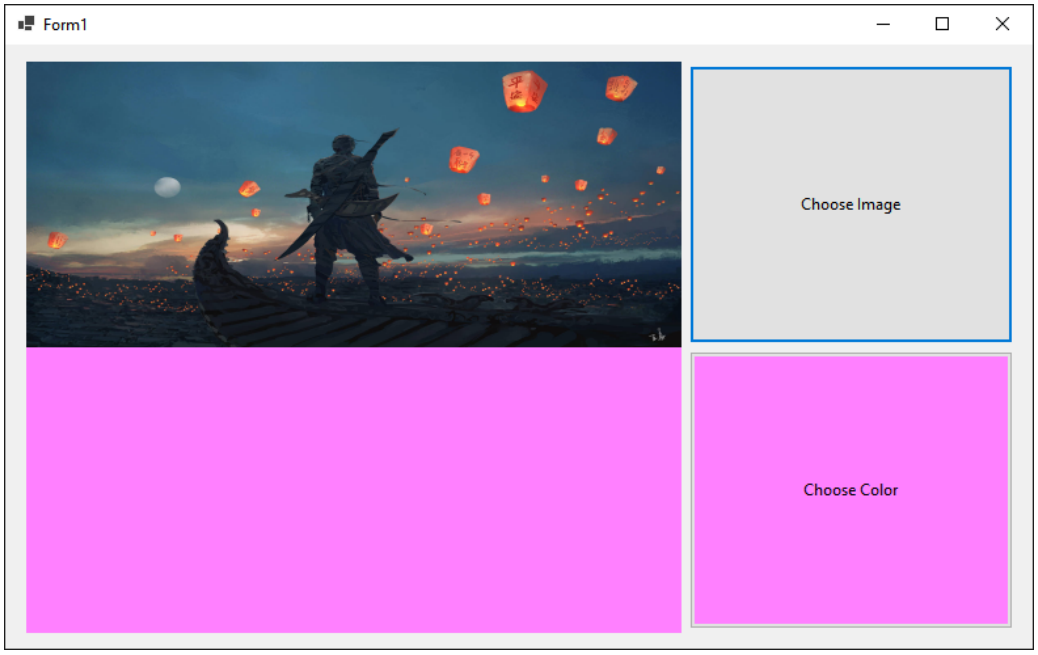


Figure 24: ImageButton logic

Using Controls – WinFormsGraphics

1. Create new project, name it "WinFormsGraphics"
2. Create layout
   - Set Form defaults
     - Set **Size** of the form to **800 x 600**
     - Set **StartPosition** to **Center Screen**
     - Set **Locked** to **True**
   - Picture Box:
     - Add a **PictureBox** to the form.
     - Rename it to **"Canvas"**.
     - Set **Anchor** to **Top, Bottom, Left, Right**
     - Set **Dock** to **Fill**
   - Prepare Drawing Area:
     - Add a **Bitmap** field to Form1:

```
1  private Bitmap drawArea;
```

     - Add following code in the Form1 constructor below `InitializeComponent()`:

```
1  drawArea = new Bitmap(Canvas.Size.Width, Canvas.Size.Height);
2  Canvas.Image = drawArea;
3  using(Graphics g = Graphics.FromImage(drawArea))
4  {
5      g.Clear(Color.LightBlue);
6  }
```

  - Alternatively, you can replace lines 3-6 with:

```
1  Graphics g = Graphics.FromImage(drawArea);
2  g.Clear(Color.LightBlue);
3  g.Dispose();
```

  - Keep in mind that all GDI+ objects **must** be disposed after usage to prevent memory leaks.

- Run the application, it should be filled with a light blue color.

Figure 25: Application Layout

3. Create some logic

   - Add **MouseDown** event to the pictureBox. It will be used to draw some circles on the bitmap.

- Add a constant field in the Form1 class:

```
private const int RADIUS = 10;
```

- Add following code to the event:

```
if(e.Button == MouseButtons.Left)
{
    using (Graphics g = Graphics.FromImage(drawArea))
    {
        g.FillEllipse(Brushes.White, e.X - RADIUS, e.Y - RADIUS, RADIUS * 2, RADIUS * 2);
    }
    Canvas.Refresh();
}
```

- Run the application and press LMB in the window. Keep in mind that `Canvas.Refresh()` is necessary in order to be able to see changes on the bitmap.

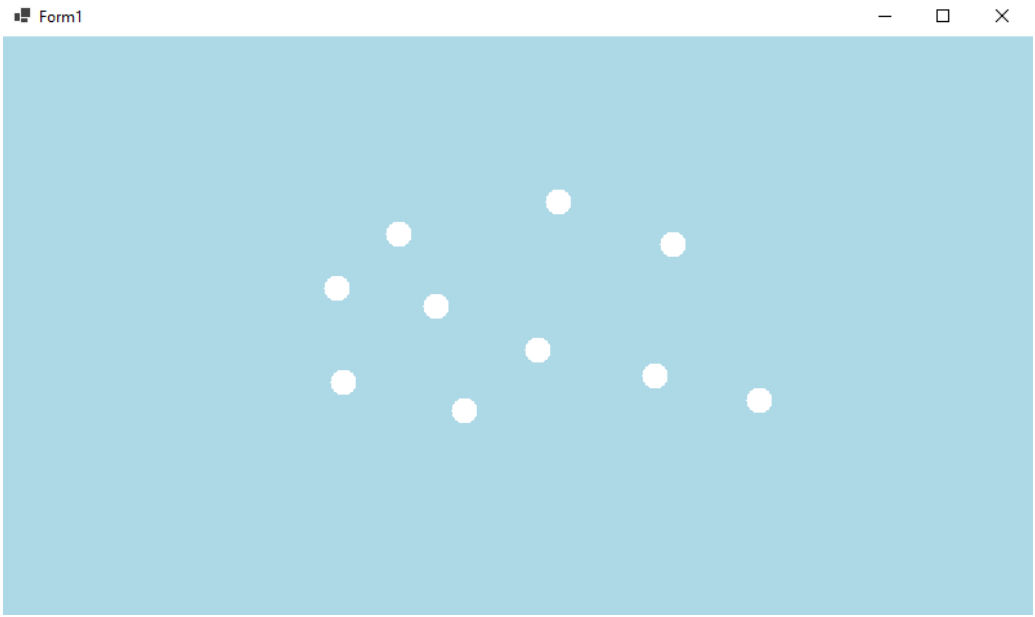


Figure 26: Logic - drawing circles

4. Add an outline to the circles

- Create a **Pen** field in the Form1 class:

```
private Pen pen;
```

- Add following code in the Form1 constructor:

```
pen = new Pen(Brushes.Black, 3);
```

- Modify the **MouseDown** event:

```
if(e.Button == MouseButtons.Left)
{
    using (Graphics g = Graphics.FromImage(drawArea))
    {
        g.FillEllipse(Brushes.White, e.X - RADIUS, e.Y - RADIUS, RADIUS * 2, RADIUS * 2);
        g.DrawEllipse(pen, e.X - RADIUS, e.Y - RADIUS, RADIUS * 2, RADIUS * 2);
    }
    Canvas.Refresh();
}
```

- Run the application and press LMB in the window. Circles should now have an outline.

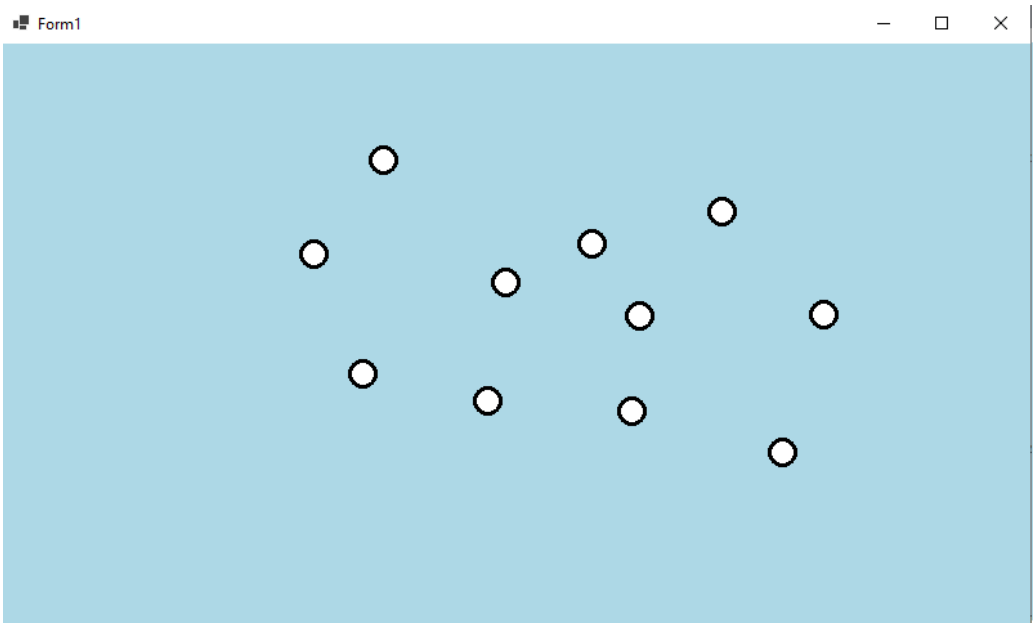


Figure 27: Logic - drawing outline

5. Drawing dashed outline

   - Create another **Pen** field in the Form1 class:

```
private Pen dashedPen;
```

   - Add following code in the Form1 constructor:

```
dashedPen = new Pen(Brushes.Black, 3);
dashedPen.DashPattern = new float[] { 2, 1 };
```

   - Add following code in the **MouseDown** event:

```
 if (e.Button == MouseButtons.Right)
{
    using (Graphics g = Graphics.FromImage(drawArea))
```

```
4       {
5           g.FillEllipse(Brushes.White, e.X - RADIUS, e.Y - RADIUS, RADIUS * 2, RADIUS * 2);
6           g.DrawEllipse(dashedPen, e.X - RADIUS, e.Y - RADIUS, RADIUS * 2, RADIUS * 2);
7       }
8       Canvas.Refresh();
9   }
```

- Run the application and press some LMB and RMB in the window. LMB draws a circle with a normal outline, RMB draws a circle with a dashed outline.

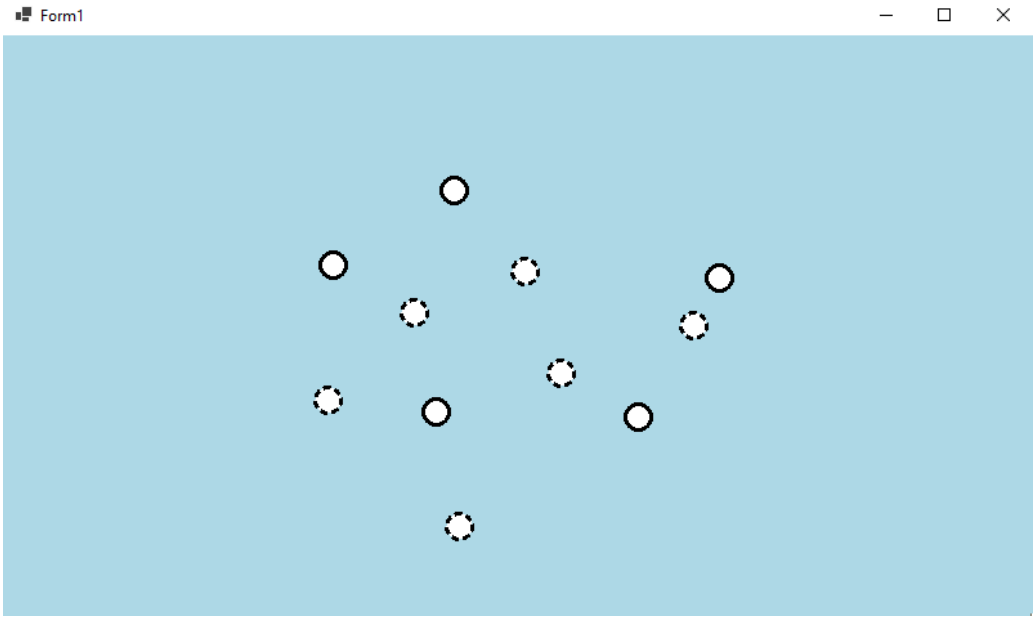


Figure 28: Logic - dashed outline