# BCIT

**Comp 3951 Topics in Application Development**

**Technical Programming Option**

# Option Head Mirela Gutica

**Winter 2022**

Mark: \_\_\_\_\_\_\_\_ /100

Lab 6

Delegates, Events and Custom and User Controls C#

This is an **individual assignment**. **No late assignments will be accepted**.

**Requirements**:

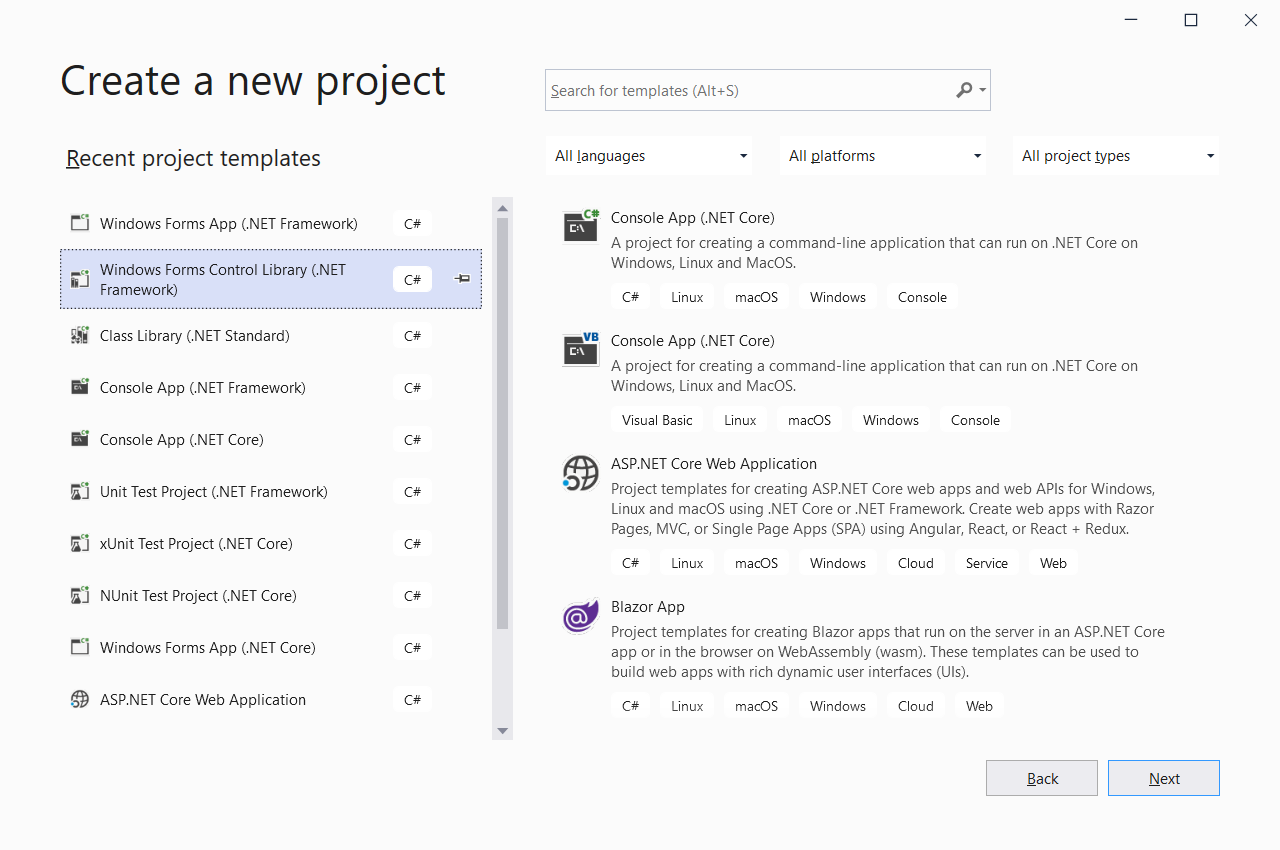
1. The code should be documented (comments) and stylish.
2. Any redundant code will be marked down.
3. You have to find solutions using the most appropriate code and functions.
4. You have to submit a .dll and a test .exe application.
5. You need to create: (1) a user control, (2) a custom control and (3) an original custom or user control of your choice.
6. Requirement 1 (25%)
   1. Create a control library (.dll). It will automatically create a **user** control.
   2. Name the .dll “your name” Control.dll (i.e. ControlLibraryPeter.dll).
   3. Create the **user controls** based on the below instructions.
   4. The name of the user control should include your name: e.g., “UserControlRGBSelectorYourName”. (i.e. UserControlRGBSelectorPeter.
   5. Test your control in a simple testing application (also submit the application). Name the test application “your name” TestApp.exe. (i.e. TestAppPeter.exe)
7. Requirement 2 (25%)
   1. Add one more control to the library: a **custom control** that extends a button.
   2. The name of the custom control should be “your name”Button (i.e. ButtonPeter).
   3. Create the custom control based on the below instructions.
   4. The button is painted with a pattern consisting of two colors and two transparencies (see below).
   5. The properties window should contain a new Category named “your name” Custom Design (i.e. Peter Custom Design).
   6. Each property should be described with a meaningful text.
   7. You have to create a new event named “your name” event (i.e. PeterEvent).
   8. The Text property should appear in your custom category. You have to describe the property text with a meaningful sentence.
   9. Your event should be fired at button click. However, the click event should still exist.
   10. Test the control in the same simple application.
8. Requirement 3 (50%):
   1. Create a **custom or** **user control** of your choice. Your control should incorporate several existing controls, a set of custom (or customized) properties, and one or more new events (that you will create). Choose a control that could be useful for your project.
   2. Add the custom or user control to the .dll.
   3. Test the control in the same simple application.

Important:

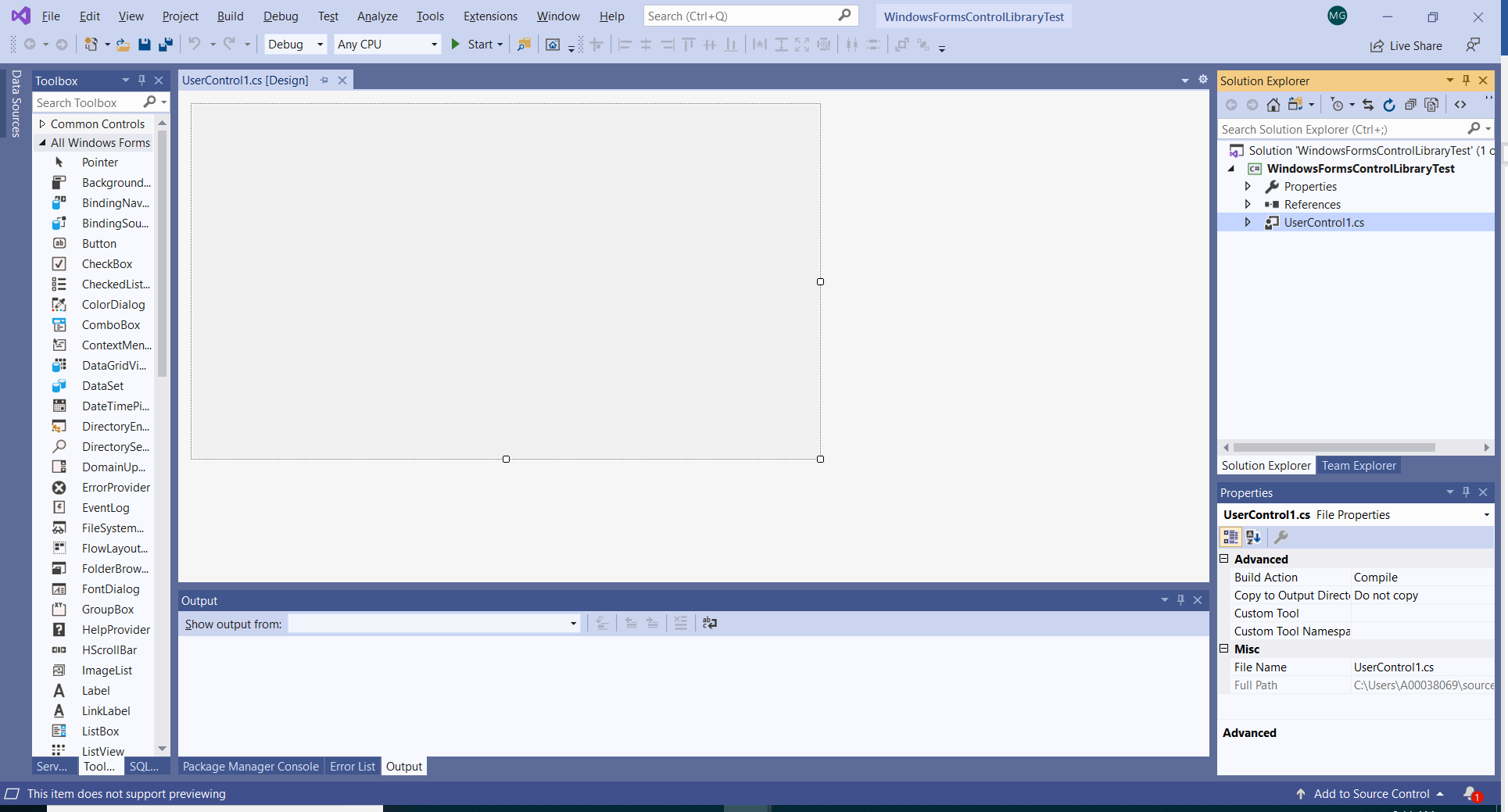
You have to submit the library and the test application projects, and the .exe and the .dll for each requirement such that I can test the functionality.

## Building a User Control

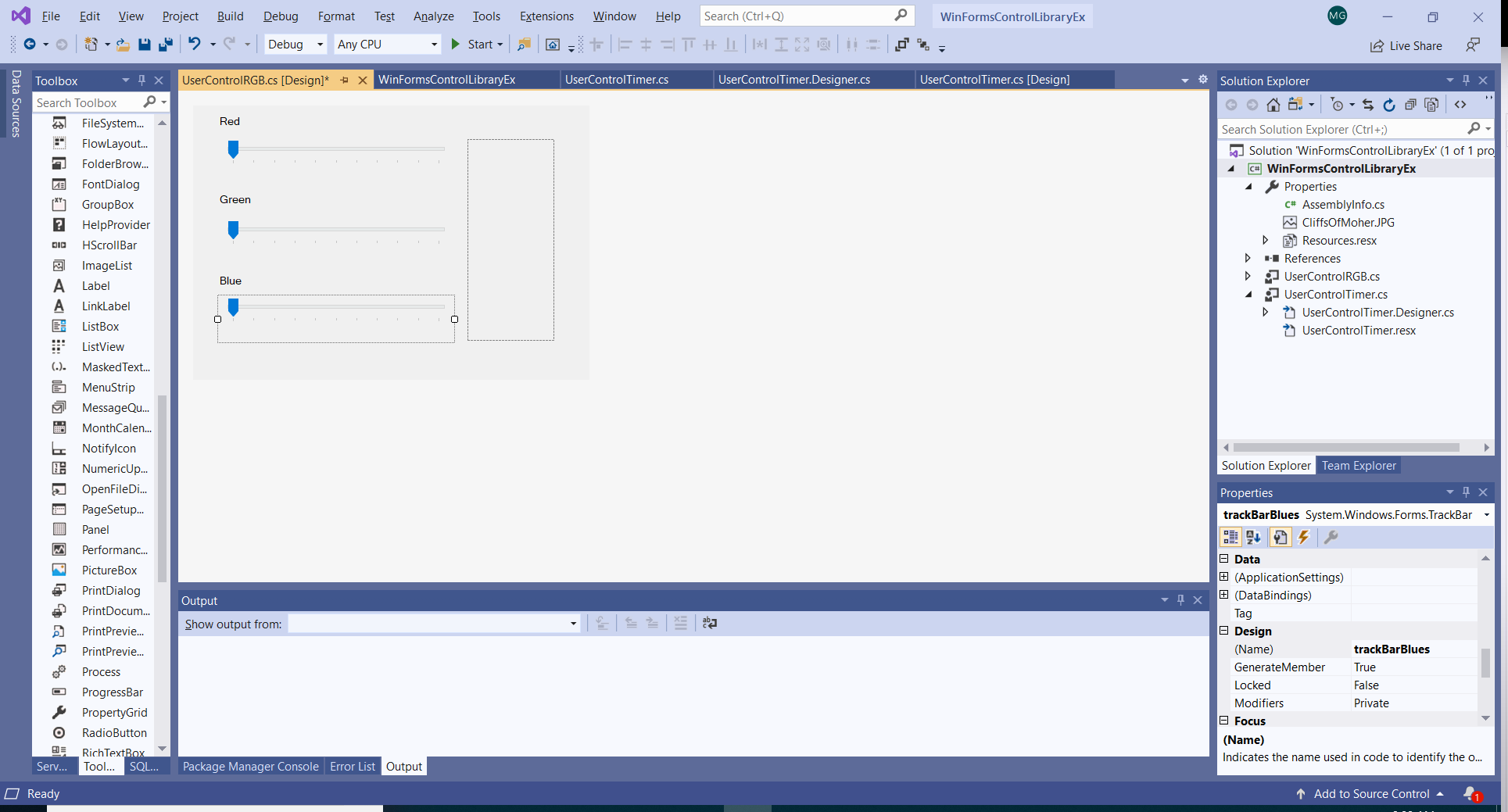
1. Open the Visual Studio and start a new project. Your project must be use the Windows **Control Library** **Project** template. Select Name your project and click OK.



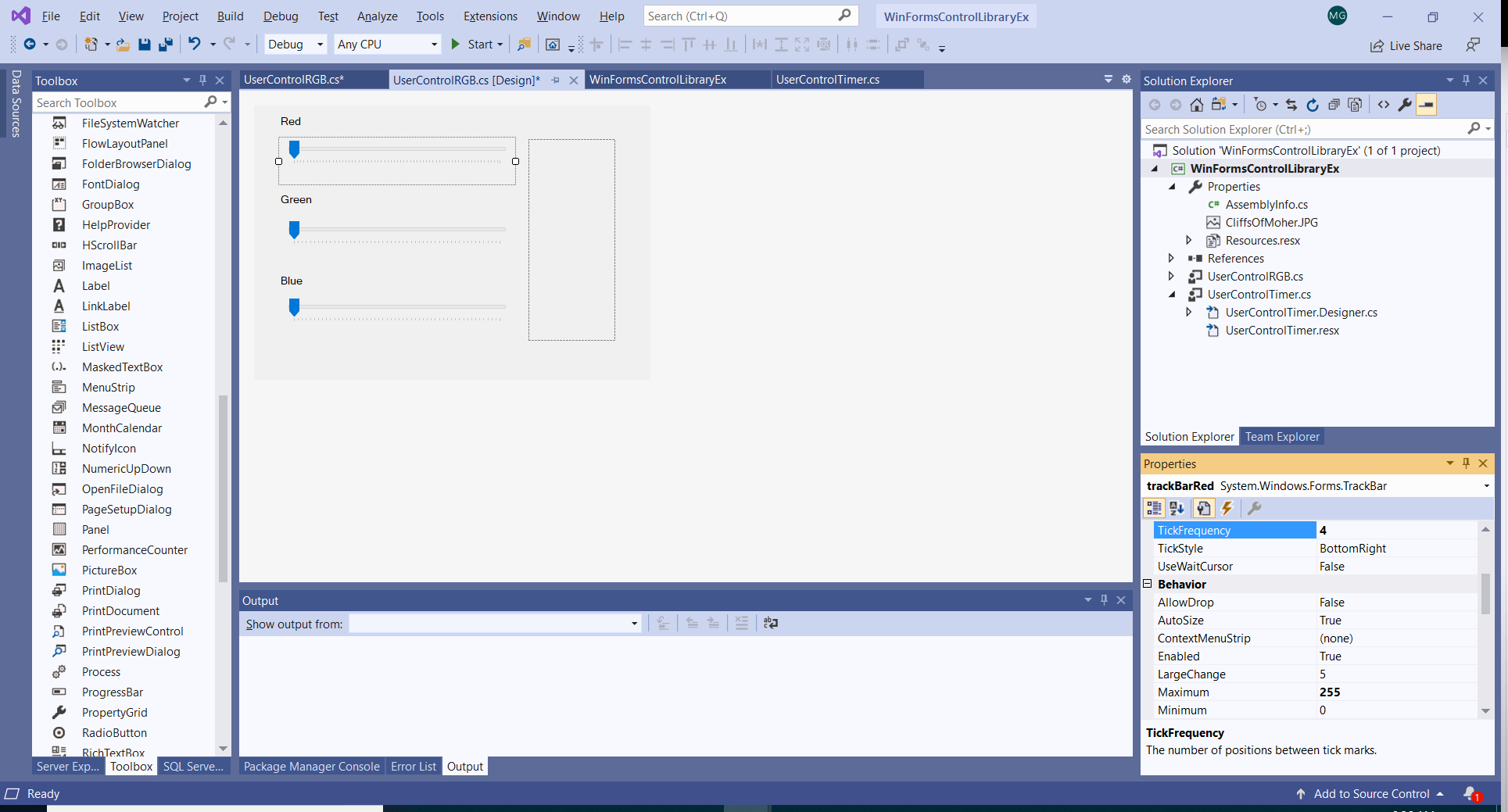
1. Once you have your project open, you will see the designer includes a ‘User Control1’. Name the user control “UserControlRGBSelectorYourName”.



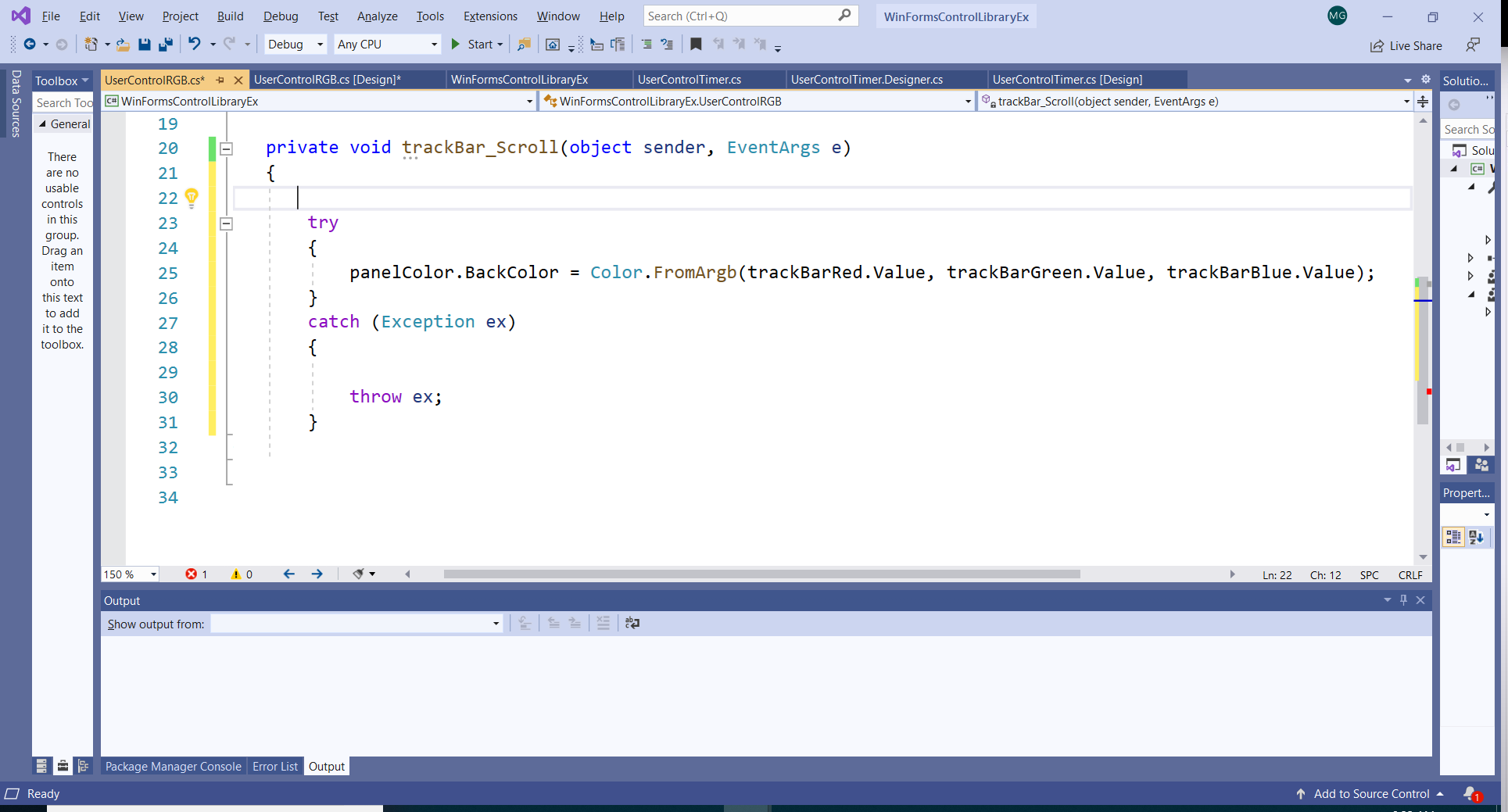
1. Build the user control using the following controls from the ToolBox: 3 labels, 3 TrackBars and one panel.



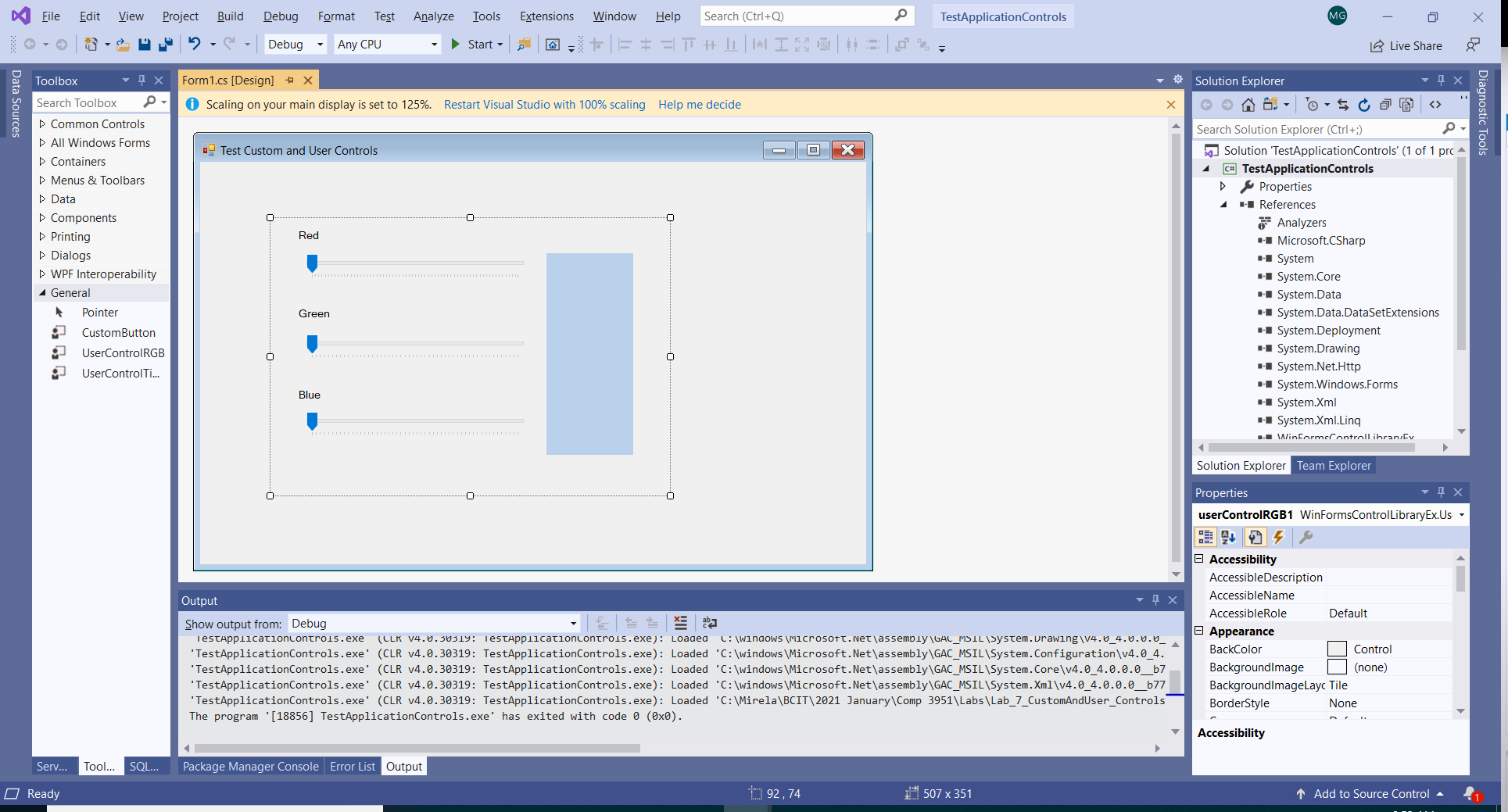
1. Name the controls properly.
2. Set the following properties for TrackBars: “Maximum” at 255, TickFrequency at 4.



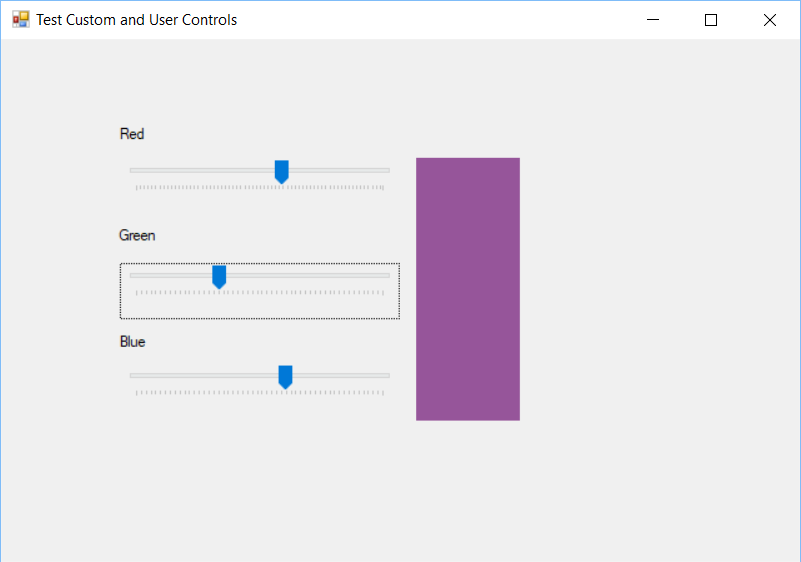
1. Create a method that displays the RGB combination of colors on the panel at the Scroll event of TaskBars. Use the same method for all three taskbars.



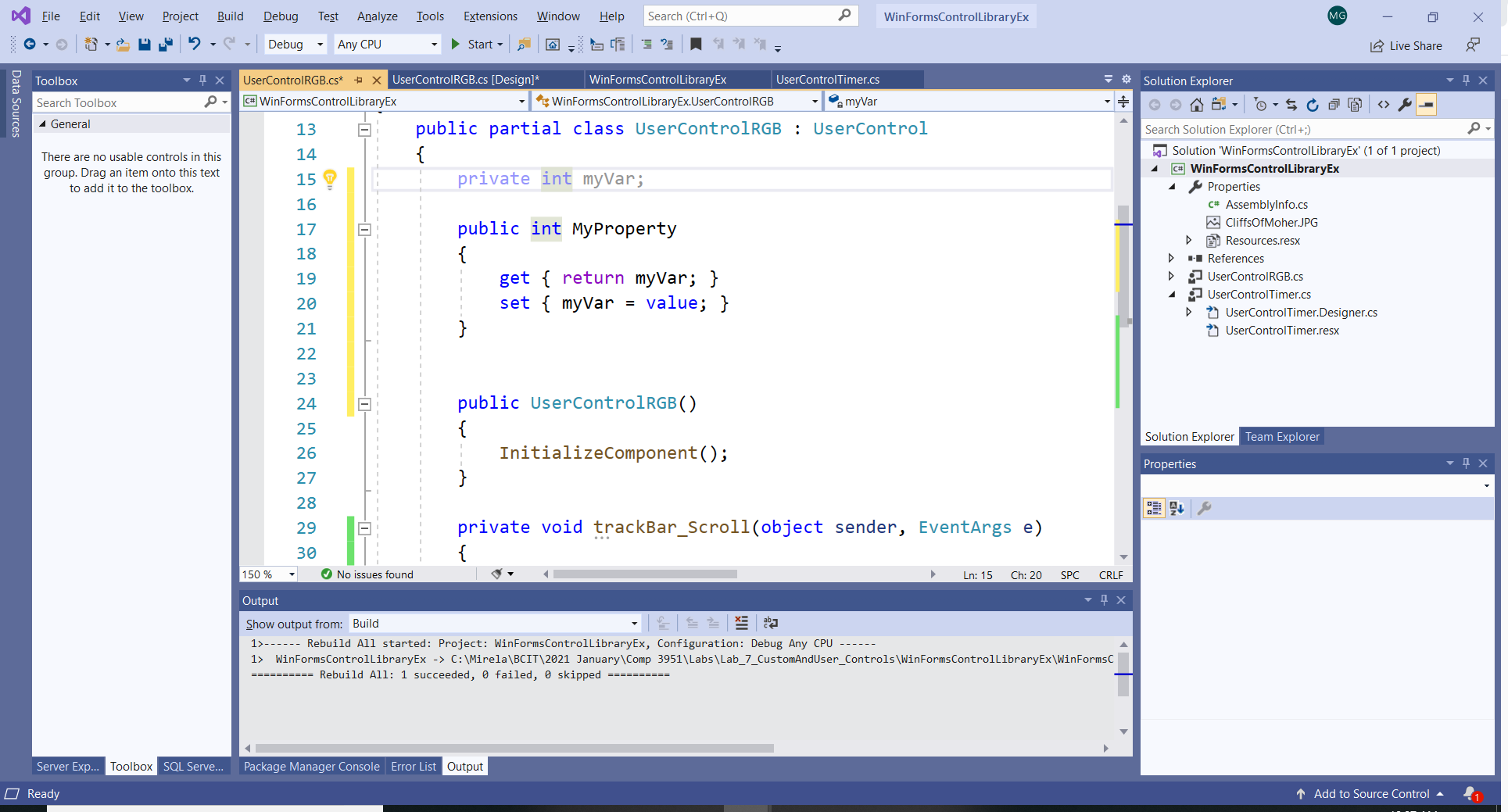
1. You can set the Back color of the panel to a color of your choice such that it is visible.
2. Build your solution. The .dll library is created.
3. *It is important to test your control. Therefore you have to create a Test application. Once the control(s) is/are added to your Test application, if you make changes to your controls, you only need to rebuild the .dll and the changes will be reflected in the Test Application.*
4. Test your control:
   1. Create a new project choosing the **Windows Forms Application** template.
   2. From a new project, we can add the compiled custom control to the toolbox by selecting Tools/Choose Toolbox Items…, clicking **Browse** and locate your .dll. The component(s) will appear in the Toolbox.

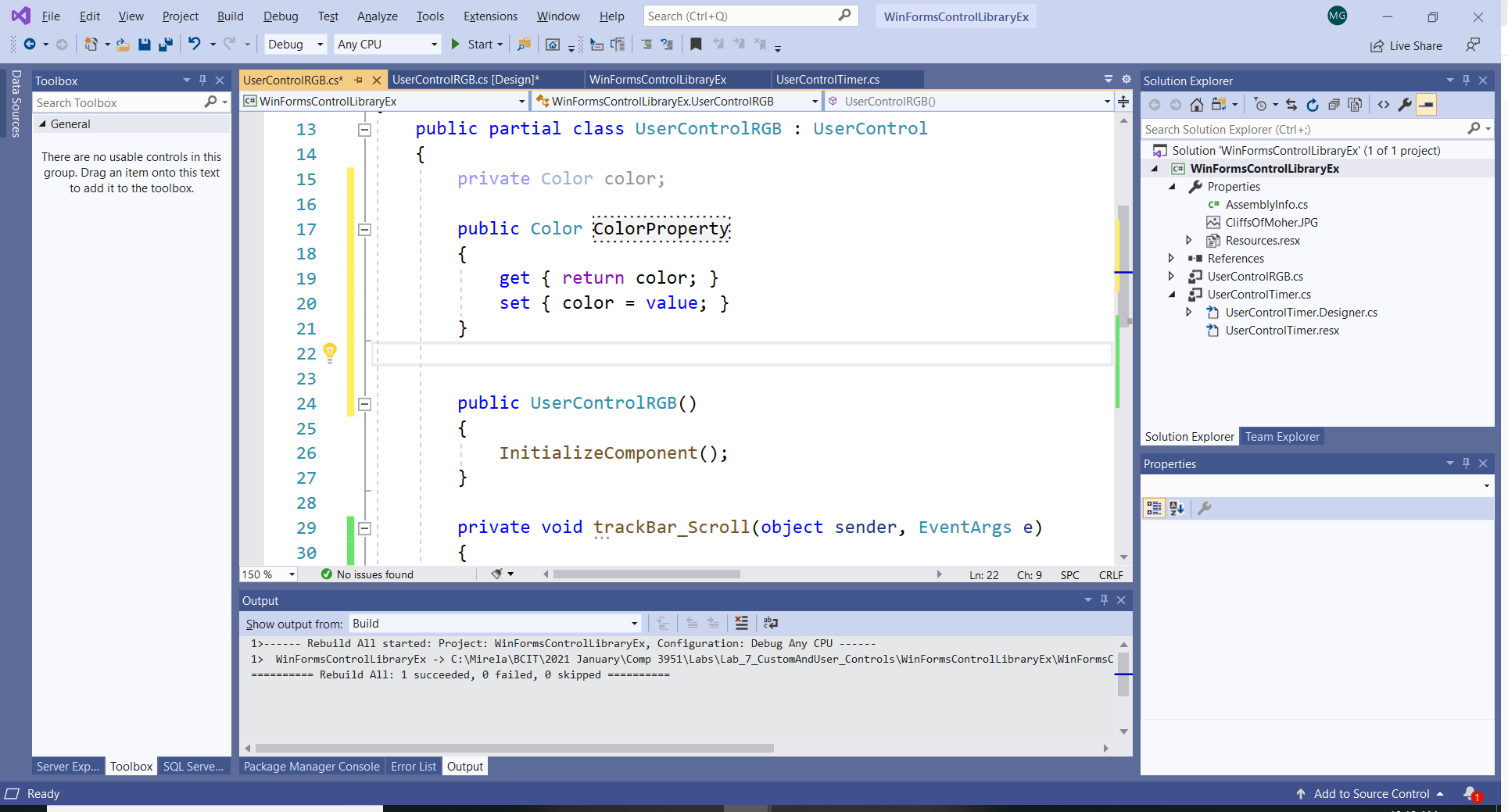


1. The functionality of your control will be like this:

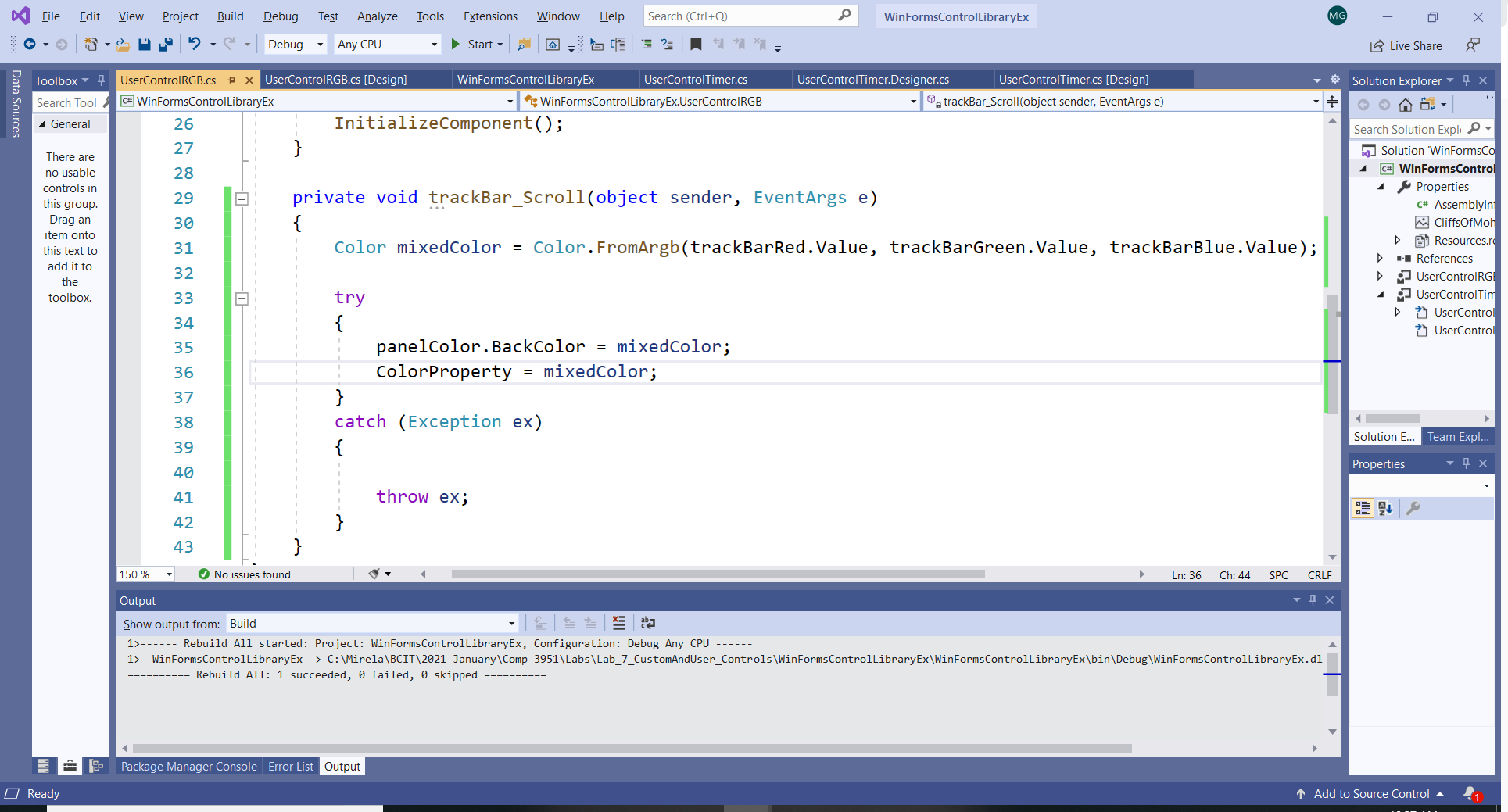


1. Add events to the RGB control.
   1. Add a field and a property to the control.

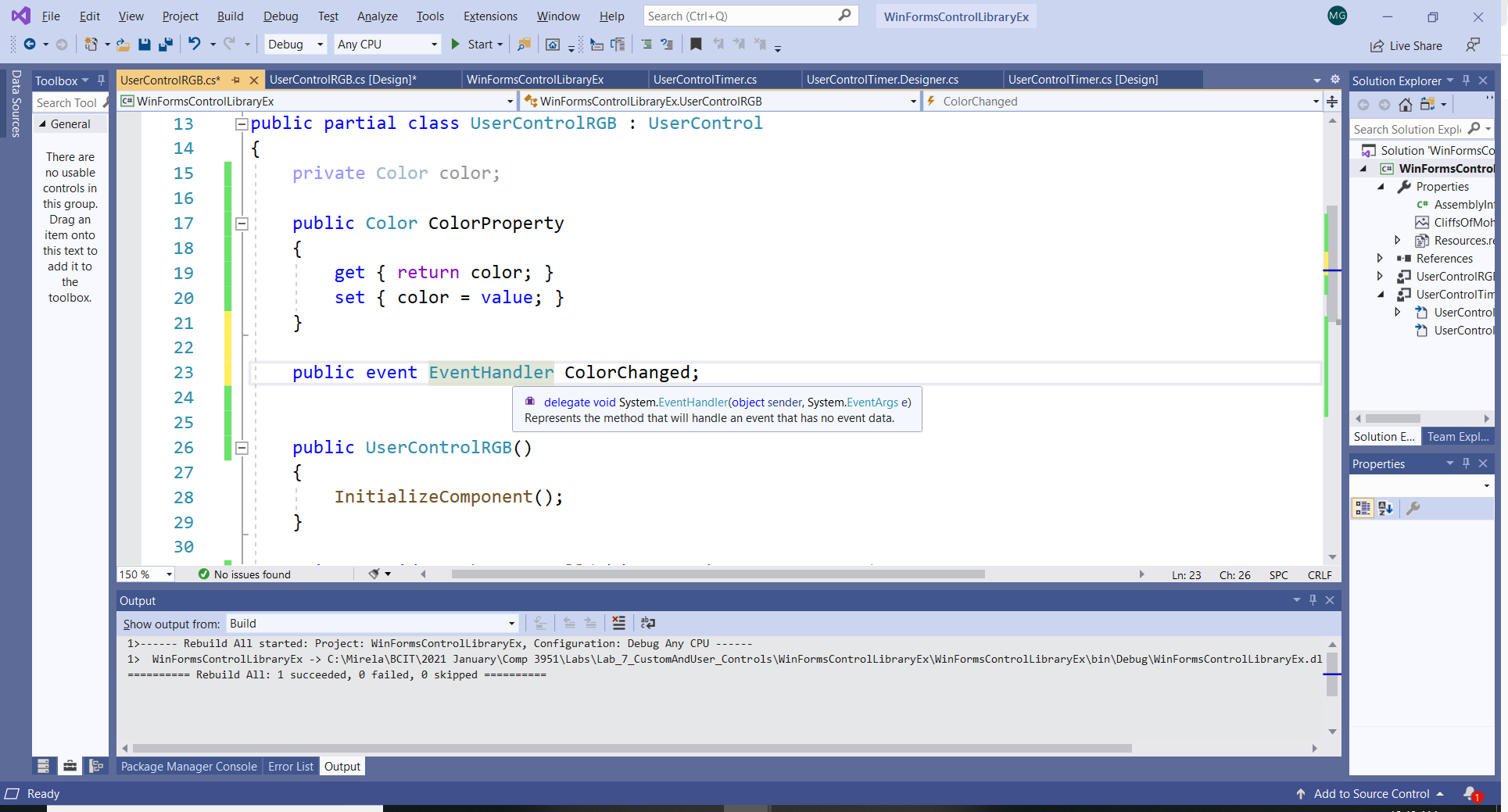




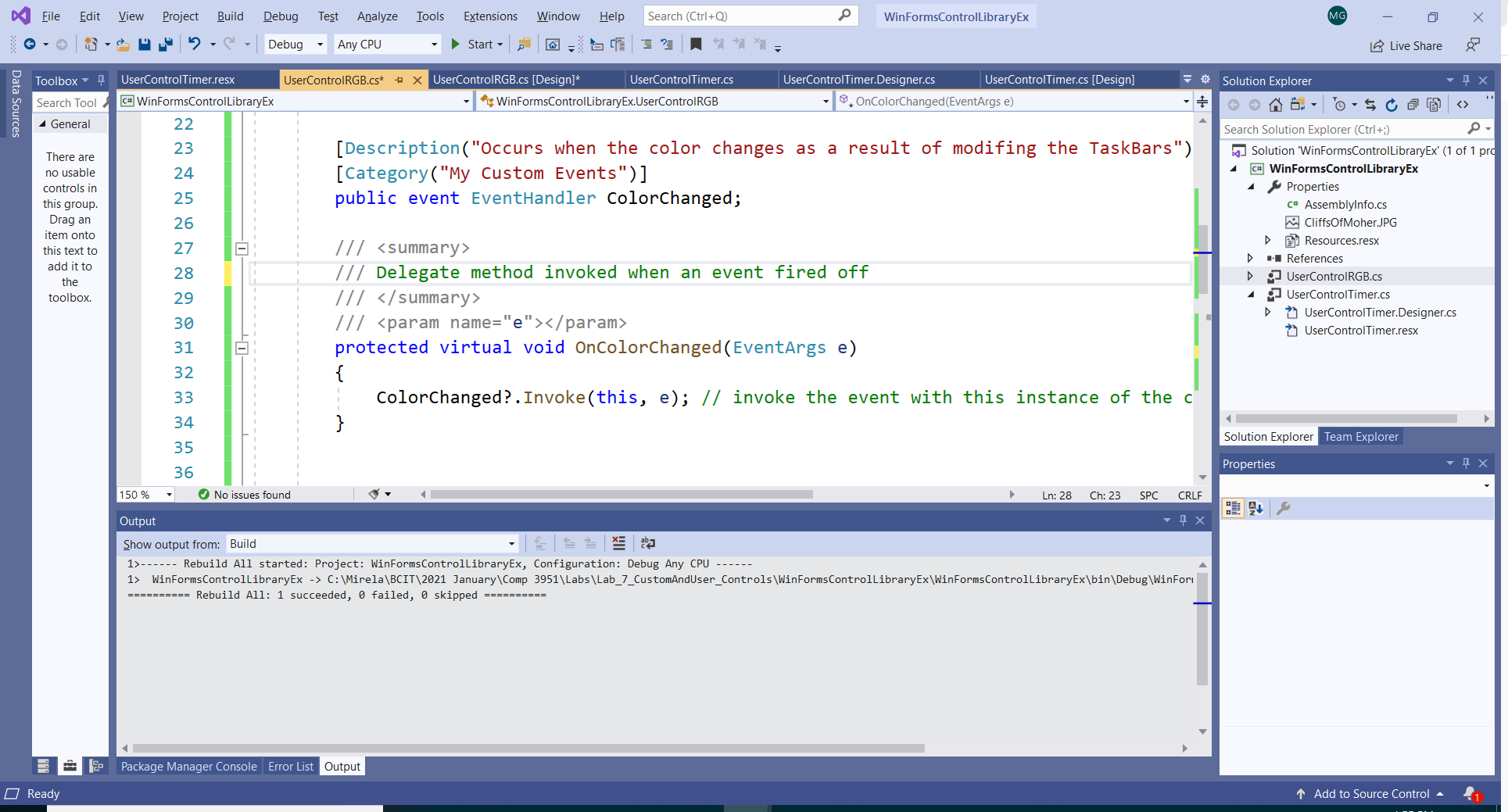
1. Update the Scroll method:



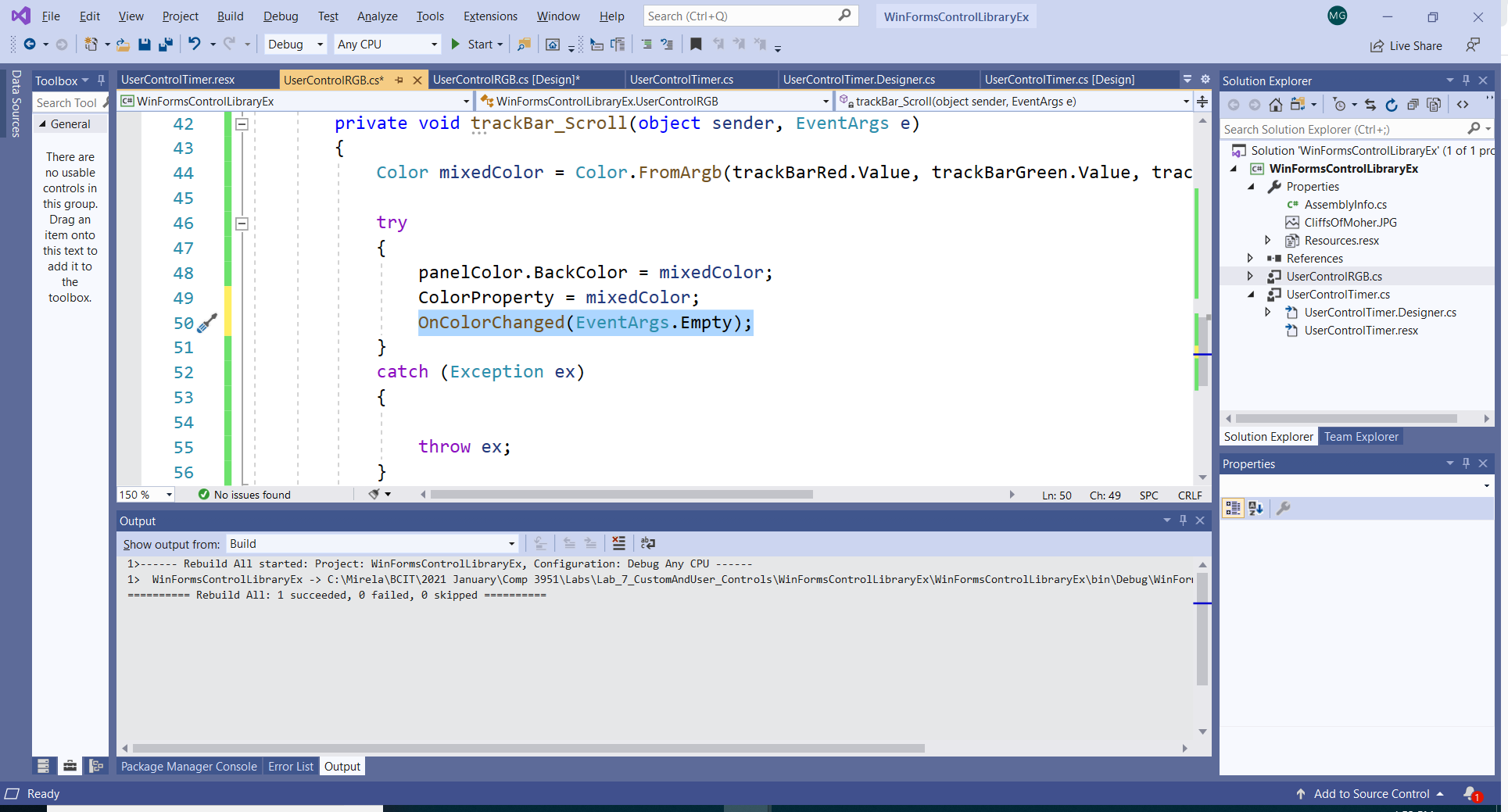
1. In the test Application notice that the UserControlRGB has now a public ColorProperty.
2. Add a public event that will allow the test application (or any other application using the control) to “listen” to the event of changing the color when the user changes the TrackBars in the control. The event declares an event with an *EventHandler* delegate. Important: the delegate already exists; no need to create a new delegate.



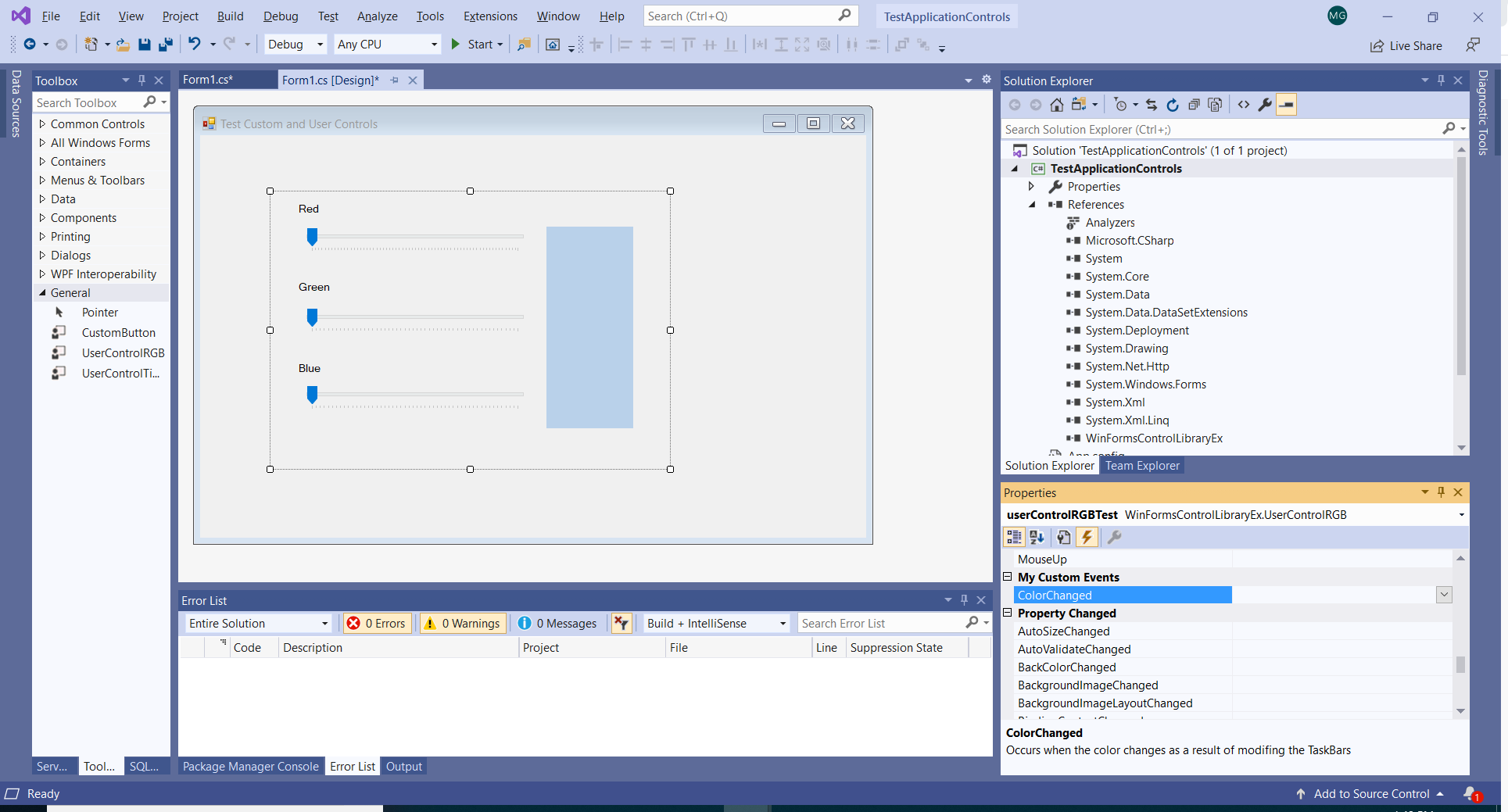
1. Add a method that is executed when an event fired off. Note the Description and Category Attributes. They will appear in the *Properties Window*.



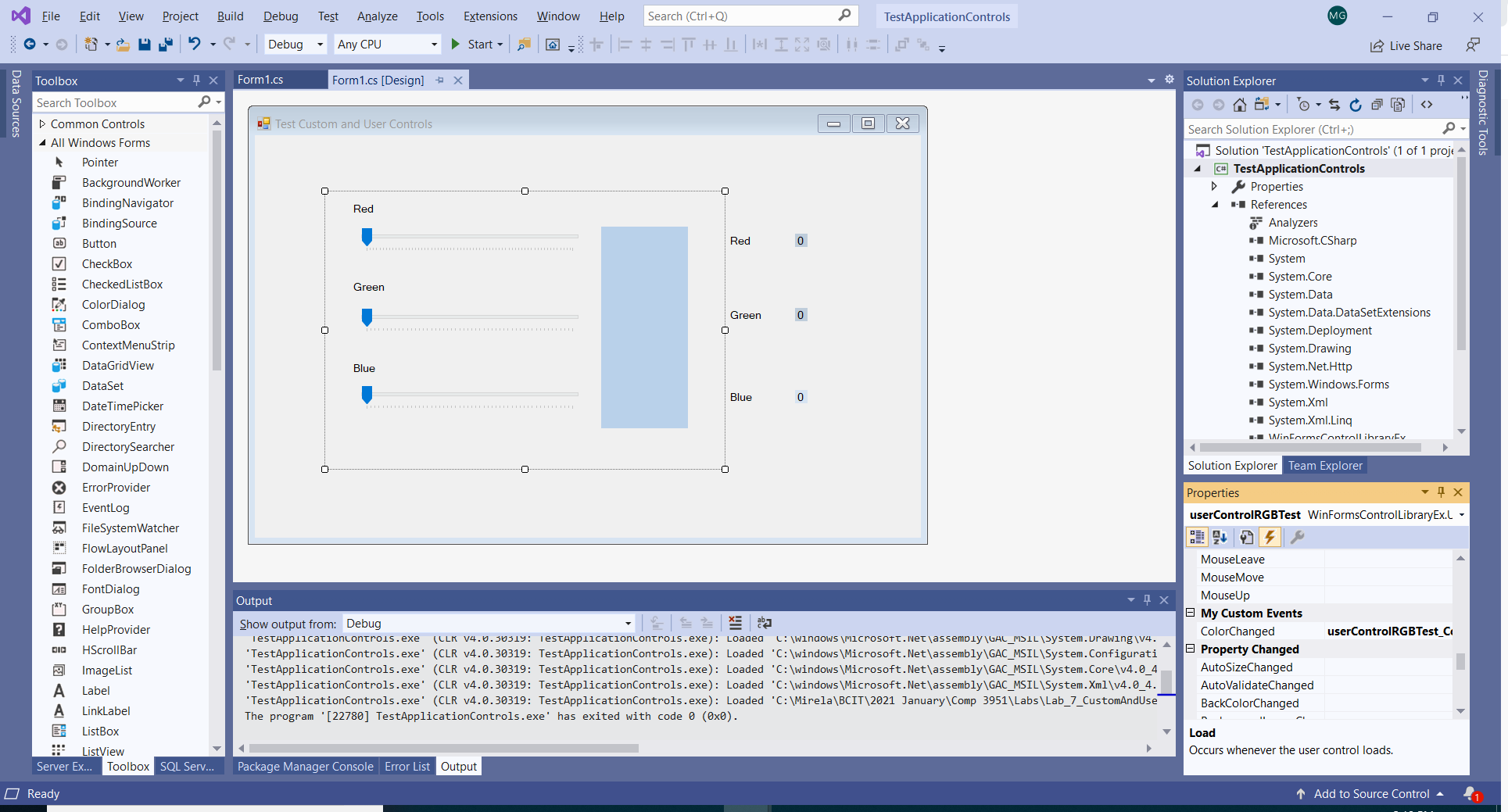
1. Call the method in the Scroll method:



1. The ColorChanged event now appears in the *Properties Window* in the Test Application:



1. In the Test application add three labels that will capture the Red, Green and Blue values:

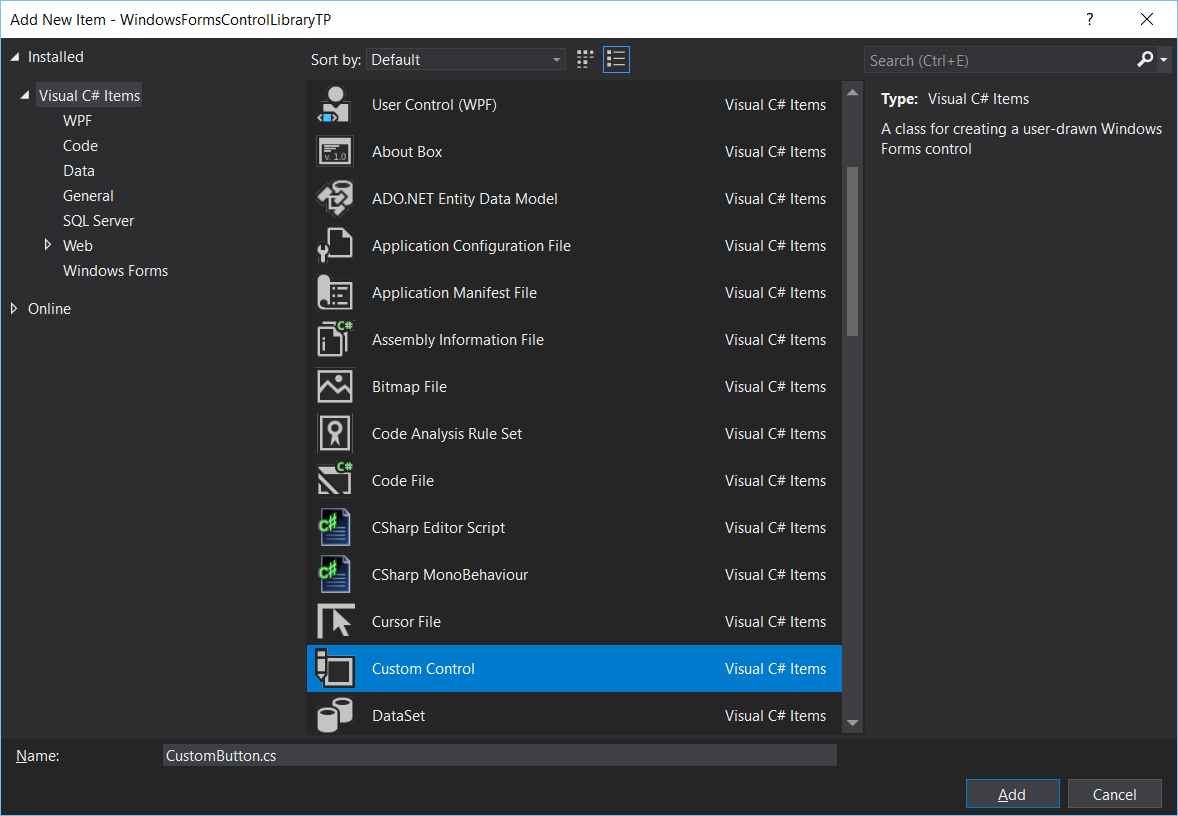


1. Implement the ColorChanged event and use the ColorProperty to extract the values of each color.

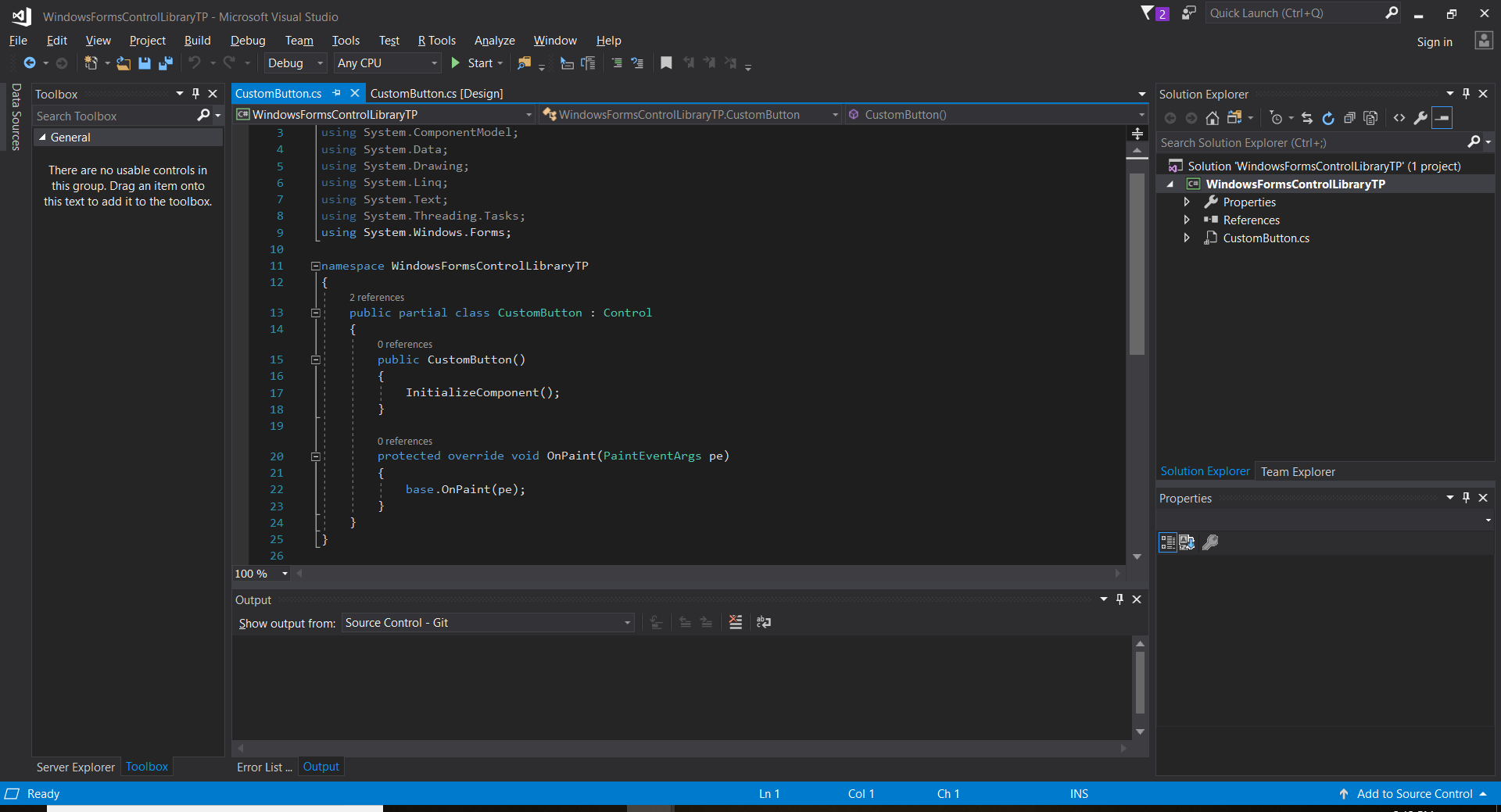
## 

## Building a Custom Control

1. In the same .dll, in the 'Project' menu select: **Project->Add New Item -> Custom Control...** and select the **Custom Control** template. 'Custom Control' is what you need in this case. Call your button CustomButton. Now click OK. A new Custom control has been added to your project.



1. There is no designer for a custom control. Switch to code view to code.
2. This is what the system created for you:



1. The first thing we must do here is to change the *base class* of the CustomButton:

Override the following line:

public partial class CustomControl : Control

with:

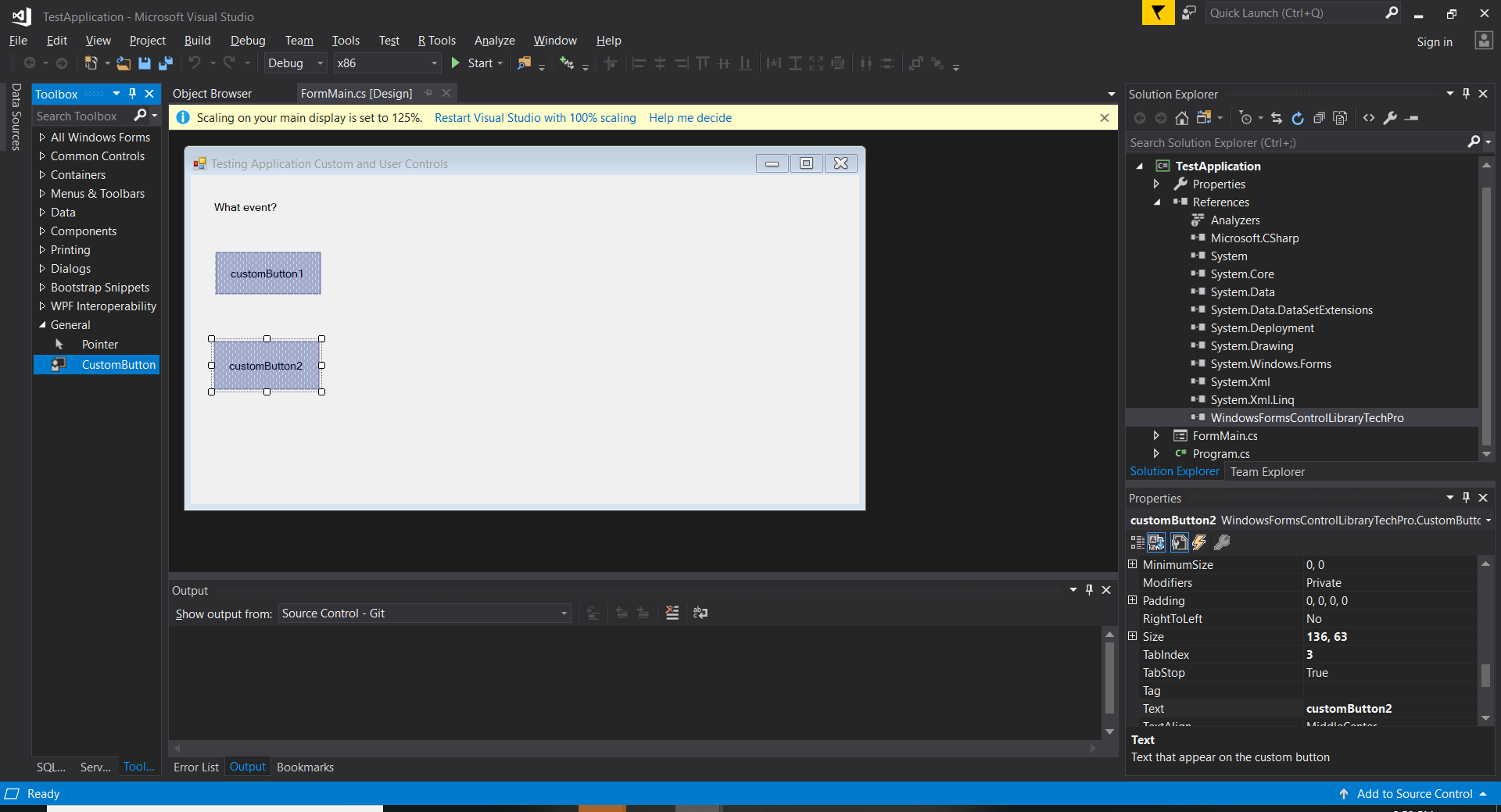
public partial class CustomButton : Button

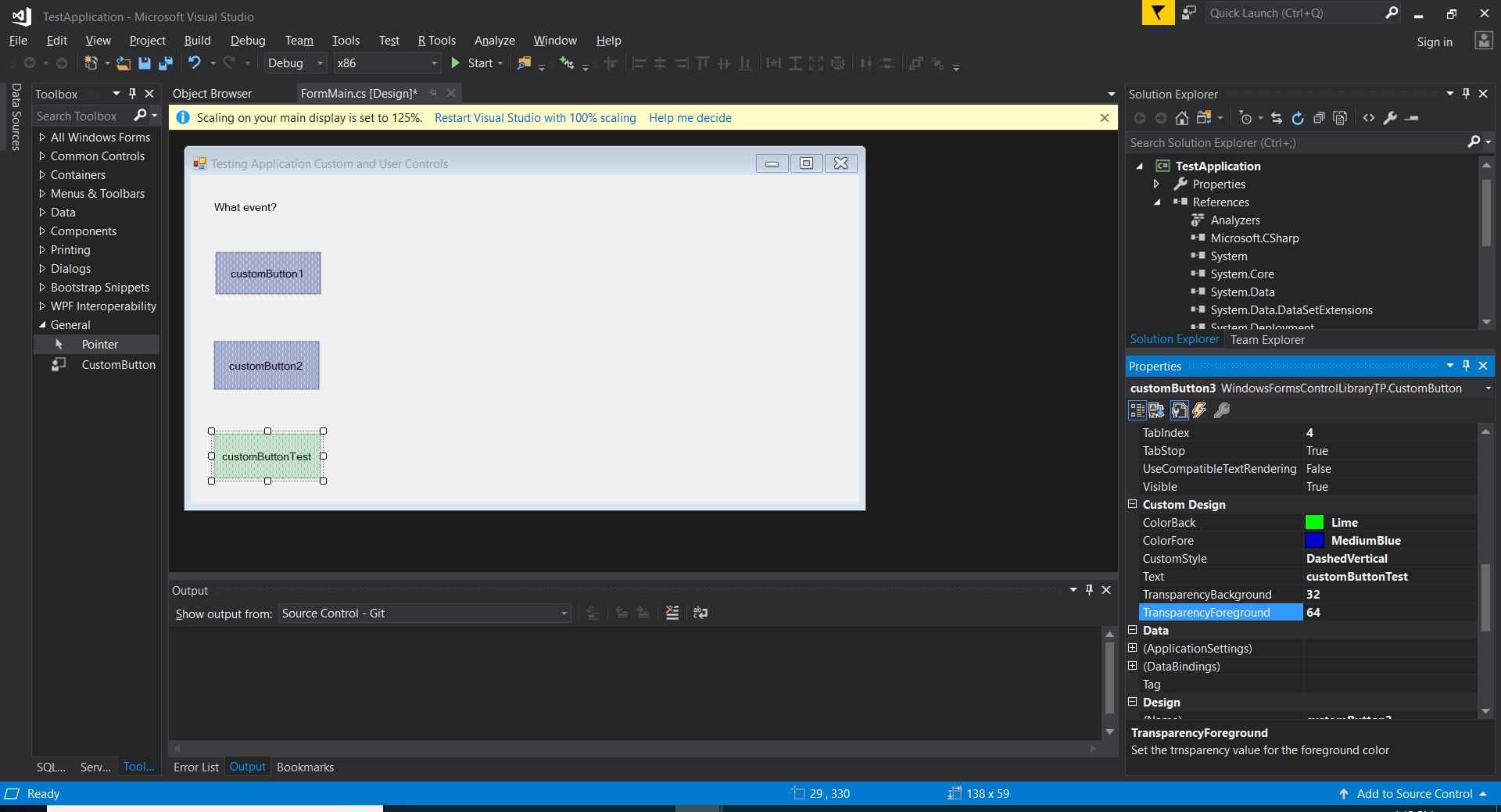
Your control is based on the System.Windows.Forms.Button class.

1. Observe the OnPaint() event: you have to modify it as this is the event that runs in the GUI for all graphics objects.
2. You need System.Drawing.Drawing2D;
3. I am giving you here the code as I believe that you will benefit and learn more from developing this part of your assignment as a tutorial. Please do not just “cut and paste”, but try to write your code.
4. Note:
   1. Properties
   2. Extending the functionality of existing properties
   3. The use of delegates
   4. Events
   5. Overriding existing events: OnMouseClick() and OnPaint()
   6. The Invalidate() function is very important as it will fire an OnPaint() event.
   7. Observe that the brush is disposed.
5. Properties and event should be published in the *Properties Window* in existing or custom categories
6. After you write the code you should build the .dll (use Build from the Debug menu).
7. Find the .dll in the debug folder.
8. Next step is to create a simple test app to test the control.

## Testing the Control

1. Create a new project choosing the **Windows Application** template.
2. From a new Windows Forms project, we can add the compiled custom control to the toolbox by selecting Tools/Choose Toolbox Items…, clicking **Browse** and locating the Control Library DLL .dll. The component **CustomButton** will appear in the Toolbox.





Code:

public partial class CustomButton : Button

{

public CustomButton()

{

InitializeComponent();

}

private Color firstColor = Color.Aqua;

private Color secondColor = Color.DarkBlue;

private int firstTransparency = 64;

private int secondTransparency = 64;

private HatchStyle customHatchStyle = HatchStyle.DashedVertical;

/// <summary>

/// declare the delegate needed for handling the custom event

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

public delegate void CustomButtonEventHandle(object sender, EventArgs e);

/// <summary>

/// add new properties for the custom button

/// </summary>

[

Category("Custom Design"),

Description("Set the pattern's style")

]

public HatchStyle CustomStyle

{

get

{

return customHatchStyle;

}

set

{

customHatchStyle = value;

Invalidate();

}

}

[

Category("Custom Design"),

Description("Set the foreground color")

]

public Color ColorFore

{

get

{

return firstColor;

}

set

{

firstColor = value;

Invalidate();

}

}

[

Category("Custom Design"),

Description("Set the background color")

]

public Color ColorBack

{

get

{

return secondColor;

}

set

{

secondColor = value;

Invalidate();

}

}

[

Category("Custom Design"),

Description("Set the trnsparency value for the foreground color")

]

public int TransparencyForeground

{

get

{

return firstTransparency;

}

set

{

firstTransparency = value;

Invalidate();

}

}

[

Category("Custom Design"),

Description("Set the trnsparency value for the background color")

]

public int TransparencyBackground

{

get

{

return secondTransparency;

}

set

{

secondTransparency = value;

Invalidate();

}

}

/// <summary>

/// override an existing property: text

/// </summary>

[

Category("Custom Design"),

Browsable(true),

Description("Text that appear on the custom button"),

DesignerSerializationVisibility(DesignerSerializationVisibility.Visible)

]

// override the text property

public override string Text

{

get

{

return base.Text;

}

set

{

base.Text = value;

}

}

/// <summary>

/// declare the event and add the event to the property window

/// </summary>

[Category("Action"),

Description("My Custom Event Click")

]

public event CustomButtonEventHandle MyEvent;

/// <summary>

/// define the custom event

/// </summary>

/// <param name="e"></param>

protected virtual void OnMyEvent(EventArgs e)

{

if (MyEvent != null)

{

MyEvent(this, e);

}

}

/// <summary>

/// override the OnMouseClick event

/// </summary>

/// <param name="e"></param>

protected override void OnMouseClick(MouseEventArgs e)

{

this.OnMyEvent(e);

this.Invalidate();

}

/// <summary>

/// override the OnPaint event because the custom control has new graphics properties;

/// OnPaint event should reflect the change

/// </summary>

/// <param name="pe"></param>

protected override void OnPaint(PaintEventArgs pe)

{

// calling the base class OnPaint event

base.OnPaint(pe);

try

{

// create the colors and the brush

Color color1 = Color.FromArgb(firstTransparency, firstColor);

Color color2 = Color.FromArgb(secondTransparency, secondColor);

HatchBrush hatchBrush = new HatchBrush(customHatchStyle, color1, color2);

// paint the button rectangle

pe.Graphics.FillRectangle(hatchBrush, ClientRectangle);

// de-allocate

hatchBrush.Dispose();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

}

## }