Making a control transparent

Asked 8 years, 4 months ago Active 2 months ago Viewed 55k times

17

12

I am currently developing a simple image editing tool using *Winforms* and `.NET 3.5` (work environment).

I have a requirement that when the user clicks a select tool button, a square (rectangle in C#) will appear that they can scale between `100x100` and `400x400`. I have this bit fixed - the issue comes with making the background of the rectangle transparent.

I'm a little unclear on if transparency is supported in `.NET 3.5`, I've tried the following:

```
SetStyle(ControlStyles.SupportsTransparentBackColor, true);
pnlSelectArea.BackColor = Color.Transparent;
pnlSelectArea.ForeColor = Color.Transparent;
selectArea1.BackColor = Color.Transparent;
selectArea1.ForeColor = Color.Transparent;
```

But this has no effect - any advice would be appreciated.

c# winforms

edited Dec 28 '15 at 7:40
Alex Jolig
9,076 17 83 125

asked Feb 20 '12 at 9:07
Adam H
1,089 3 14 26

Check this stackoverflow.com/questions/72994/... – papaiatis Feb 20 '12 at 9:10

Thanks for the help - I can't apply any of this to my solution but I appreciate the effort. – Adam H Feb 20 '12 at 10:17

You can find my simple approach explained in the post here: – aleksandaril Dec 21 '15 at 21:28

5 Answers

Active | Oldest | Votes

46

This is my special Control which contains an opacity property, it 100% works:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.ComponentModel.Design;
using System.Drawing;
using System.Windows.Forms;
using System.Windows.Forms.Design;
```

```
public bool enab = false;
private int m_opacity = 100;

private int alpha;
public TranspCtrl()
{
    SetStyle(ControlStyles.SupportsTransparentBackColor, true);
    SetStyle(ControlStyles.Opaque, true);
    this.BackColor = Color.Transparent;
}

public int Opacity
{
    get
    {
        if (m_opacity > 100)
        {
            m_opacity = 100;
        }
        else if (m_opacity < 1)
        {
            m_opacity = 1;
        }
        return this.m_opacity;
    }
    set
    {
        this.m_opacity = value;
        if (this.Parent != null)
        {
            Parent.Invalidate(this.Bounds, true);
        }
    }
}

protected override CreateParams CreateParams
{
    get
    {
        CreateParams cp = base.CreateParams;
        cp.ExStyle = cp.ExStyle | 0x20;
        return cp;
    }
}

protected override void OnPaint(PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Rectangle bounds = new Rectangle(0, 0, this.Width - 1, this.Height - 1);

    Color frmColor = this.Parent.BackColor;
    Brush bckColor = default(Brush);

    alpha = (m_opacity * 255) / 100;

    if (drag)
    {
        Color dragBckColor = default(Color);

        if (BackColor != Color.Transparent)
        {
            int Rb = BackColor.R * alpha / 255 + frmColor.R * (255 - alpha) /
```

```
                int Bb = BackColor.B * alpha / 255 + frmColor.B * (255 - alpha) /
255;
                dragBckColor = Color.FromArgb(Rb, Gb, Bb);
            }
            else
            {
                dragBckColor = frmColor;
            }

            alpha = 255;
            bckColor = new SolidBrush(Color.FromArgb(alpha, dragBckColor));
        }
        else
        {
            bckColor = new SolidBrush(Color.FromArgb(alpha, this.BackColor));
        }

        if (this.BackColor != Color.Transparent | drag)
        {
            g.FillRectangle(bckColor, bounds);
        }

        bckColor.Dispose();
        g.Dispose();
        base.OnPaint(e);
    }

    protected override void OnBackColorChanged(EventArgs e)
    {
        if (this.Parent != null)
        {
            Parent.Invalidate(this.Bounds, true);
        }
        base.OnBackColorChanged(e);
    }

    protected override void OnParentBackColorChanged(EventArgs e)
    {
        this.Invalidate();
        base.OnParentBackColorChanged(e);
    }
}
```

edited Apr 28 '16 at 14:53

sirdank
2,333 2 19 41

answered Feb 20 '12 at 10:31

Amen Ayach
3,873 1 19 23

Worked nicely for me. – Rob Aug 27 '14 at 9:06

ppl just copy and paste in your custom class and change constructor name. works perfectly!! thank you – hasan Aug 8 '15 at 15:46

3 This currently has a serious bug. In the OnPaint override, it is calling Dispose on the e.Graphics object, which should not be done. The application is likely to crash if you dispose of the Graphics object that was passed in. If you didn't create it, you shouldn't dispose of it. – Tim May 18 '16 at 22:01

Works nicely when "Control" was changed to "Panel". – Kaitlyn Jul 13 '16 at 21:48

@Amen Ayach: The application will crash if i set SetStyle(ControlStyles.AllPaintingInWmPaint, true) to

4

You will need to use `Opacity` property and set it to zero to make form invisible.

If you want to make a control Transparent, as you have tried in your example, See this article

How to: Give Your Control a Transparent Background

It say the code you have written, must be in constructor of the control. Hence, I guess, you will need to create a custom control derived from your `pnlSelectArea` 's type most probaably a button. In in that custom control's constructor you can write code to set its style and color.

edited Feb 20 '12 at 10:41

answered Feb 20 '12 at 9:13

Maheep
5,003 2 20 44

I don't want to make the control invisible. I want to make its contents invisible so I can use it to mouse over different parts of the image and use it as a 'select tool' – Adam H Feb 20 '12 at 10:16

Then set those content's Opacity. – Maheep Feb 20 '12 at 10:20

1 Brother Mahepp no Opacity property in winforms just in WPF – Amen Ayach Feb 20 '12 at 10:29

It is available for Form in winforms as well. OP can use a borderless Form for his/her usage. – Maheep Feb 21 '12 at 4:03

@AmenAyach msdn.microsoft.com/en-us/library/… – Maheep Feb 21 '12 at 4:04

1

Here is what worked for me with because the other solutions did not work.

This is with transparent UserControl added to ListView/TreeView Control Collection

I know it says ButtonRenderer but it should work for any controls.

In the UserControl:

```
protected override void OnPaint(PaintEventArgs e)
{
    ButtonRenderer.DrawParentBackground(e.Graphics, this.ClientRectangle, this);
}
```

in the Parent control:

```
protected override void WndProc(ref Message m)
{
    if(m.Msg == 0xF)
        foreach(Control c in this.Controls) { c.Invalidate(); c.Update(); }

    base.WndProc(ref m);
}
```

edited Dec 15 '15 at 3:57

answered Dec 15 '15 at 2:28

0

great!! I finally managed to draw transparent shapes. I've added a virtual method

```
Draw(g);
```

right before

```
bckColor.Dispose();
g.Dispose();
base.OnPaint(e);
```

and at the end the declaration of the virtual method

```
protected virtual void Draw(Graphics g){ }
```

Now I can continue creating my own Transparent shapes, graphics etc ...

answered Oct 13 '15 at 13:14

Agguro

329 2 11

Have you ever putted your custom control on top of other controls? – Dominota May 18 at 15:30

0

There is one simple workaround for this. You can create an image with a transparent background (PNG) and add it for the Image property of the icon. This works fine as information does not have much flexibility in styling. Sometime this might not be suitable for everyone. Remember this is **only a workaround.**

PS: Add where ever the text on the image and keep blank for the text property.

answered Apr 11 at 19:25

Sanke

333 1 5 22