C# 系统应用之透明罩MyOpaqueLayer实现360界面阴影效果

原创 Eastmount 最后发布于2014-03-10 14:33:56 阅读数 6091 ☆ 收藏 展开

在完成“个人电脑使用记录清除软件”中,我设计的winform界面需要应用到类似于"360安全卫士"的透明罩效果,文章主要引述了如何使用自定义组件MyOpaqueLayer,并自定义类OpaqueCommand中定义显示透明罩函数ShowOpaqueLayer和隐藏透明罩函数HideOpaqueLayer实现,同时如何对控件添加透明罩及遇到的问题.

一.自定义透明罩MyOpaqueLayer组件

(声明:此段代码引用自海华博客http://www.cnblogs.com/JuneZhang/archive/2012/07/06/2579215.html)

在添加透明罩控件\组件时,我的想法是"右键项目->添加->添加控件",但添加没有成功,在网上也没有讲到该添加的基础方法,由于以前也没遇到过自定义控件的问题,所以只好采取的方法是拖拽修改过MyOpaqueLayer.cs文件至项目中,具体代码如下(含详细注释):

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

namespace MyOpaqueLayer
{
    /*
     * [ToolboxBitmap(typeof(MyOpaqueLayer))]
     * 用于指定当把你做好的自定义控件添加到工具栏时,工具栏显示的图标。
     * 正确写法应该是
     * [ToolboxBitmap(typeof(XXXXControl),"xxx.bmp")]
     * 其中XXXXControl是你的自定义控件，"xxx.bmp"是你要用的图标名称。
    */
    [ToolboxBitmap(typeof(MyOpaqueLayer))]

    /// <summary>
    /// 自定义控件:透明罩控件(继承Control)
    /// </summary>
    public class MyOpaqueLayer : System.Windows.Forms.Control
    {
        private bool _transparentBG = true;       //是否使用透明
        private int _alpha = 125;                 //设置透明度

        private System.ComponentModel.Container components = new System.ComponentModel.Container();
        public MyOpaqueLayer()
            : this(125, true)
        {
        }

        public MyOpaqueLayer(int Alpha, bool IsShowLoadingImage)
        {
            SetStyle(System.Windows.Forms.ControlStyles.Opaque, true);  //设置控件样式
            base.CreateControl();                                       //创建控件
            this._alpha = Alpha;
            //放置加载进度的图片代码此处被省略
        }
```

```
//释放组件占用内存
                        protected override void Dispose(bool disposing)
{
    if (disposing)
    {
        if (!((components == null)))
        {
            components.Dispose();
        }
    }
    base.Dispose(disposing);
}

/// <summary>
/// 自定义绘制窗体
/// </summary>
/// <param name="e"></param>
protected override void OnPaint(System.Windows.Forms.PaintEventArgs e)
{
    float vlblControlWidth;
    float vlblControlHeight;

    Pen labelBorderPen;                     //定义Pen
    SolidBrush labelBackColorBrush;         //定义单色画笔

    if (_transparentBG)                     //使用透明
    {
        Color drawColor = Color.FromArgb(this._alpha, this.BackColor);
        labelBorderPen = new Pen(drawColor, 0);
        labelBackColorBrush = new SolidBrush(drawColor);
    }
    else
    {
        labelBorderPen = new Pen(this.BackColor, 0);
        labelBackColorBrush = new SolidBrush(this.BackColor);
    }
    base.OnPaint(e);
    vlblControlWidth = this.Size.Width;
    vlblControlHeight = this.Size.Height;
    e.Graphics.DrawRectangle(labelBorderPen, 0, 0, vlblControlWidth, vlblControlHeight);
    e.Graphics.FillRectangle(labelBackColorBrush, 0, 0, vlblControlWidth, vlblControlHeight);
}

//获取创建控件句柄时所需要的创建参数
protected override CreateParams CreateParams //v1.10
{
    get
    {
        CreateParams cp = base.CreateParams;  //扩展派生类CreateParams属性
        cp.ExStyle |= 0x00000020;             //开启WS_EX_TRANSPARENT,使控件支持透明
        return cp;
    }
}

/*
 * [Category("myOpaqueLayer"), Description("是否使用透明,默认为True")]
 * 一般用于说明你自定义控件的属性（Property）
 * Category用于说明该属性属于哪个分类,Description自然就是该属性的含义解释。
 */
[Category("MyOpaqueLayer"), Description("是否使用透明,默认为True")]
public bool TransparentBG
{
```

```
            get
                        {
                return _transparentBG;
            }
            set
            {
                _transparentBG = value;
                this.Invalidate();
            }
        }
        //设置透明度
        [Category("MyOpaqueLayer"), Description("设置透明度")]
        public int Alpha
        {
            get
            {
                return _alpha;
            }
            set
            {
                _alpha = value;
                this.Invalidate();
            }
        }

        //初始化窗体
        private void InitializeComponent()
        {
            this.SuspendLayout();      //临时挂起控件的布局逻辑,它与ResumeLayout()配合使用
            this.ResumeLayout(false);  //恢复正常逻辑
        }
    }
}
```

二.自定义透明罩类OpaqueCommand

在第一部分我们已经自定义透明罩组件,此时需要自定义类OpaqueCommand并调用其方法ShowOpaqueLayer(显示遮罩层)和HideOpaqueLayer(隐藏遮罩层).可以在“解决方法”中右键项目名->添加->类,具体代码如下:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace EMSecure
{
    class OpaqueCommand
    {
        //透明罩
        private MyOpaqueLayer.MyOpaqueLayer m_OpaqueLayer = null;

        /// <summary>
        /// 显示透明层
        /// </summary>
        /// <param name="control">控件</param>
        /// <param name="alpha">透明度</param>
        /// <param name="isShowLoadingImage">是否显示图标</param>
```

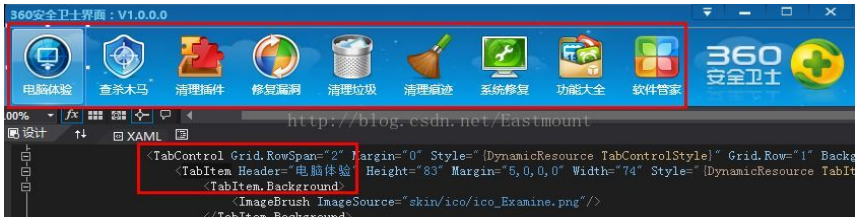
```
        public void ShowOpaqueLayer(Control control, int alpha, bool isShowLoadingImage)          {
            try
            {
                if (this.m_OpaqueLayer == null)
                {
                    this.m_OpaqueLayer = new MyOpaqueLayer.MyOpaqueLayer(alpha, isShowLoadingImage);
                    control.Controls.Add(this.m_OpaqueLayer);
                    this.m_OpaqueLayer.Dock = DockStyle.Fill;
                    this.m_OpaqueLayer.BringToFront();
                }
                this.m_OpaqueLayer.Enabled = true;
                this.m_OpaqueLayer.Visible = true;
            }
            catch (Exception msg)              //异常处理
            {
                MessageBox.Show(msg.Message);
            }
        }

        /// <summary>
        /// 隐藏透明层
        /// </summary>
        public void HideOpaqueLayer()
        {
            try
            {
                if (this.m_OpaqueLayer != null)
                {
                    this.m_OpaqueLayer.Visible = false;
                    this.m_OpaqueLayer.Enabled = false;
                }
            }
            catch (Exception msg)             //异常处理
            {
                MessageBox.Show(msg.Message);
            }
        }
    }
}
```

三.使用透明罩

在定义透明罩控件和类后,如何实现该界面的效果,我推荐的方法是:

1.设置toolBar控件,在items(集合)中添加相应的图标\文字构成不同的ToolItem,每次透明罩遮掩不同的Item即可,如果是wfp使用TabControl\TabItem.但由于toolBar被toolStrip替代,不太会使用该控件,但仍然推荐该方法(有的没有定义透明罩控件,而是通过3张透明程度不同的图,设置可见属性实现该效果).

运行结果:



2.在代码设计器Form中添加MyOpaqueLayer控件,此时就能看见6个透明罩的MyOpaqueLayer控件,该方法不需要设置自定义类OpaqueCommand调用其方法,而是设置如下代码:

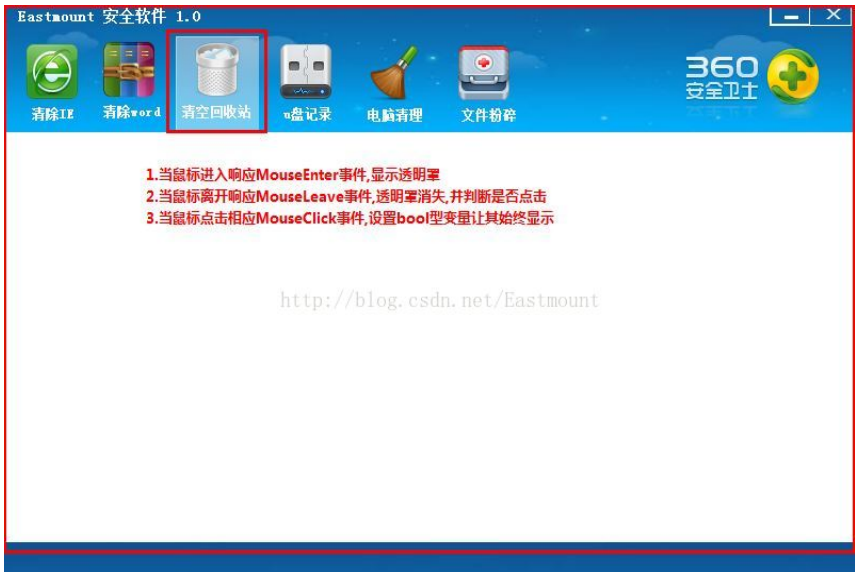
```
//鼠标离开
private void myOpaqueLayer1_MouseLeave(object sender, EventArgs e)
```

```
{
      this.myOpaqueLayer1.Visible = false;
    this.label4.ForeColor = Color.White;
}
//鼠标进入
private void pictureBox1_MouseEnter(object sender, EventArgs e)
{
    this.myOpaqueLayer1.Visible = true;
    this.label4.ForeColor = Color.Yellow;
}
```

设计器中form如下图所示:

但由于不知道如何添加该控件拖动至设计器中,所以我采取的方法是

3.自定义6个panel,通过鼠标事件进入panelmol1_MouseEnter(object sender, EventArgs e)\鼠标离开事件panelmol1_MouseLeave(object sender, EventArgs e)\鼠标点击事件panelmol1_MouseClick(object sender, EventArgs e)实现,最后我的运行结果如下图所示:



下面只给出使用透明罩控件和类的基本调用代码并省略Click部分(因为做的不是很好),请读者体会与自己设计:

```
//自定义类OpaqueCommand
OpaqueCommand cmd = new OpaqueCommand();
//定义点击panel时透明罩情况
bool isClick = false;

//鼠标进入"清除IE"
private void panel_mol1_MouseEnter(object sender, EventArgs e)
{
    //透明罩设置 没点击才取消透明罩
    cmd.ShowOpaqueLayer(panel_mol1, 125, true);
}
private void panel_mol1_MouseEnter(object sender, EventArgs e)
{
    cmd.HideOpaqueLayer();
}
```

四.总结

个人感觉该设计还有很多地方需要自己改进,同时想实现ToolBar方法和了解如何添加自定义组件而不是拖拽,期望自己能解决这些问题.希望该文章对大家有所帮助,同时感谢下面3篇文章及博主,是讲述透明罩和如何自定义控件的文章,个人感觉非常不错.如果文章中有不足或错误的地方,请海涵!

(By:Eastmount 2014-3-10 中午2点 原创:http://blog.csdn.net/eastmount)

参考资料及在线笔记:

1.C#实现Winform自定义半透明遮罩层-海华(主要参考透明罩的设定)

http://www.cnblogs.com/JuneZhang/archive/2012/07/06/2579215.html

2.C#中自定义控件-杨友山(自定义控件简述)

http://blog.csdn.net/yysyangyangyangshan/article/details/7078471

3.C#自定义控件开发-百度文库(详细介绍自定义控件\复合控件\扩展控件)

http://wenku.baidu.com/view/89a47f6e58fafab069dc02bf.html