ComponentOne

Imaging for Windows Phone

Corporate Headquarters
ComponentOne LLC
201 South Highland Avenue
3rd Floor
Pittsburgh, PA 15206 · USA

Internet: **info@ComponentOne.com**

Web site: **http://www.componentone.com**

Sales

E-mail: sales@componentone.com

Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)

This manual was produced using ComponentOne Doc-To-Help™.

Table of Contents

ComponentOne Imaging for Windows Phone Overview

Put powerful imaging capabilities at your fingertips with **ComponentOne Imaging™ for Windows Phone**. Quickly load and edit images (PNG and JPEG) and save them back, or display animated GIFs.

Help with ComponentOne Studio for Windows Phone

Getting Started

For information on installing **ComponentOne Studio for Windows Phone**, licensing, technical support, namespaces and creating a project with the control, please visit Getting Started with Studio for Windows Phone.

What's New

For a list of the latest features added to **ComponentOne Studio for Windows Phone**, visit What's New in Studio for Windows Phone.

ComponentOne Imaging for Windows Phone Assemblies and Controls

The **C1.Phone.Imaging.dll** assembly includes controls that enhance Windows Phone imaging functionality.

Main Classes

The following main classes are included in the **C1.Phone.Imaging.dll** assembly:

- **C1GifImage:** Similar to the regular **System.Windows.Media.Imaging.BitmapImage** class, but with support for animated GIF images. To use this class, simply assign it to the **Source** property of an **Image** or **C1Image** element.
- **C1Bitmap:** Provides programmatic creation of images, and importing/exporting from PNG, JPG, and GIF.

Imaging for Windows Phone Key Features

Make the most of **Imaging for Windows Phone** by taking advantage of the following key features:

- **Edit Images Programmatically**

  The C1Bitmap class allows you to access individual pixels to create special effects, crop, resize, or transform images in any way you want.

- **Reduce/Crop Images**

  Editing the pixels enables you to resize images and reduce the resolution, which reduces the file size and results in faster upload time. Also, you can crop users' images in order to upload only part of them as you do in Facebook or any other web user account.

- **Support for Animated GIF Files**

The C1GifImage class enables you to add animated GIF files to your Windows Phone applications. The standard Image control only supports PNG and JPEG formats. Animated GIFs are compact and allow you to add attractive visual elements to your applications with minimal effort.

- **Play, Pause, and Stop Animated GIFs**

  The C1GifImage class provides media player-like commands and allows you to control the GIF animations programmatically. You can use these methods to animate GIFs while performing a task, creating interesting progress indicators, or simply better integrating the animations with the state of the application.

Working with C1Bitmap

C1Bitmap allows you to create images programmatically and provides importing/exporting from PNG, JPG, and GIF. You can crop images and reduce file size for faster upload time, or warp images – just for fun!

In the following topics, we'll refer to the **Imaging for Windows Phone manipulation** sample in the **ControlExplorer** sample installed with the product. The **ControlExplorer** sample is located in the **C:\Documents and Settings\<username>\My Documents\ComponentOne Samples\Studio for Windows Phone** or **C:\Users\<username>\Documents\ComponentOne Samples\Studio for Windows Phone** folder.

Opening and Saving the Image

Two vital operations in our application are the open and save actions. We have configured the open image method to work for both when the user drops an image file onto the application and if the user decides to browse their machine. We accomplish this by passing a simple stream as our input, and we configure the application to load an image from the stream. Once an image is loaded, we display it in a standard **Image** control on the page, but behind the scenes we are loading this into a C1Bitmap component. From there we will be able to further manipulate the image with actions such as cropping and warping.

For example, the following code, taken from the **manipulation** sample for **Imaging for Windows Phone** in the **ControlExplorer**, allows you to choose a photo on the phone and load the image.

```
private void btnPickPicture_Click(object sender, RoutedEventArgs e)
        {
            Microsoft.Phone.Tasks.PhotoChooserTask photoChooser = new
Microsoft.Phone.Tasks.PhotoChooserTask();
            photoChooser.ShowCamera = true;
            photoChooser.Completed += (s2, e2) =>
            {
                if (e2.Error == null)
                {
                    GoToState(ImageManipulationStep.PickedPhoto,
e2.ChosenPhoto);
                }
                else
                {
                    // error picking photo
                    MessageBox.Show("Error picking photo.\n\nPlease
disconnect the device from the computer, if connected.");
                    GoToState(ImageManipulationStep.Start);
                }
            };
            photoChooser.Show();
        }

        void LoadImageStream(Stream stream)
        {
            bitmap.SetStream(stream);
            if (bitmap.Width * bitmap.Height > imageSize)
            {
                var scale = Math.Sqrt(imageSize * 1.0 / (bitmap.Width *
bitmap.Height));
                var resizedBitmap = new C1Bitmap((int)(bitmap.Width *
scale), (int)(bitmap.Height * scale));
                resizedBitmap.Copy(bitmap, true);
```

```
                bitmap = resizedBitmap;
            }
            originalBitmap.Copy(bitmap, false);
            screen.Copy(bitmap, false);

            image.Source = screen.ImageSource;
            image.Width = screen.Width;
            image.Height = screen.Height;

            // clear selection
            selection = new Rect(0, 0, bitmap.Width, bitmap.Height);
            UpdateMask();
        }
```

Note: The **ControlExplorer** sample is located in the *C:\Documents and Settings\<username>\My Documents\ComponentOne Samples\Studio for Windows Phone* or *C:\Users\<username>\Documents\ComponentOne Samples\Studio for Windows Phone* folder.

Printing the Image

Once the image is loaded we can easily print. Silverlight 4's printing capabilities add plenty of value to almost any Silverlight application. The printing features basically include a **PrintDocument** component. We will be modeling our printing code after the same sample from Microsoft used for drag-and-drop. To print we simply call the **PrintDocument.Print** method, and in the **PrintPage** event we set the document's **PageVisual** property to any UI Element we want, in this case it's our image container.

```
PrintDocument printDocument = new PrintDocument();
private void btnPrint_Click(object sender, RoutedEventArgs e)
{
    printDocument.Print("My Image");
}

void printDocument_PrintPage(object sender, PrintPageEventArgs e)
{
    e.PageVisual = imageGrid;
    e.HasMorePages = false;
}
```

Cropping an Image

Being able to crop an image entirely on the client is a highly useful task. Thankfully, with C1Bitmap or the **WriteableBitmap** class (introduced in Silverlight 3) this is achievable in Silverlight and Windows Phone. The C1Bitmap component provides an API that is easier to work with when doing any bitmap related manipulation, primarily because it can get and set simple colors and it gives more direct access to pixels with the **GetPixel** and **SetPixel** methods.

If you take a look at the **manipulation** sample for **Imaging for Windows Phone** in the **ControlExplorer**, you'll see an example of how to make a selection and crop an image.

Note: The **ControlExplorer** sample is located in the *C:\Documents and Settings\<username>\My Documents\ComponentOne Samples\Studio for Windows Phone* or *C:\Users\<username>\Documents\ComponentOne Samples\Studio for Windows Phone* folder.

Here is the XAML that defines the elements needed to create our crop box. It consists of four shaded rectangles which mask the regions that will be cropped out.

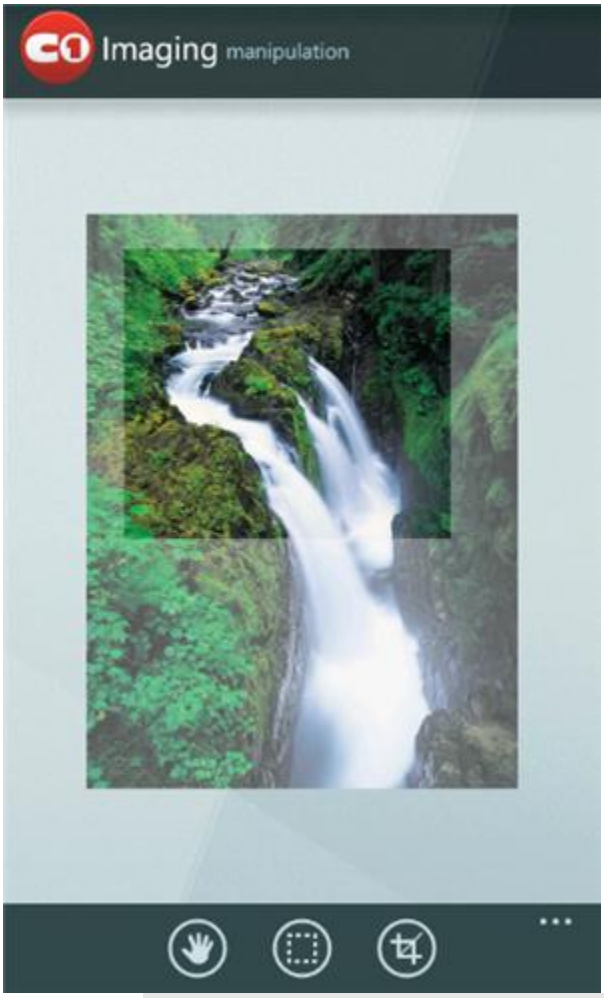
```
            <Grid x:Name="topMask" Grid.ColumnSpan="2"
Background="{StaticResource MaskBrush}" />
```

```
            <Grid x:Name="bottomMask" Grid.Column="1" Grid.Row="2"
Grid.ColumnSpan="2" Background="{StaticResource MaskBrush}" />
            <Grid x:Name="leftMask" Grid.RowSpan="2" Grid.Row="1"
Background="{StaticResource MaskBrush}" />
            <Grid x:Name="rightMask" Grid.Column="2" Grid.RowSpan="2"
Background="{StaticResource MaskBrush}" />
```

The code needed to manipulate the crop box is quite complex. The purpose of the crop box UI is to generate a simple Rect which will be used by the C1Bitmap to determine the coordinates of the cropping. By tapping the **Select** button on the application toolbar the user can drag to display a crop box over the area to be cropped. As the user drags the corners of the selection box, the **C1.Phone.C1GestureListener** is used to capture the vertical and horizontal change.

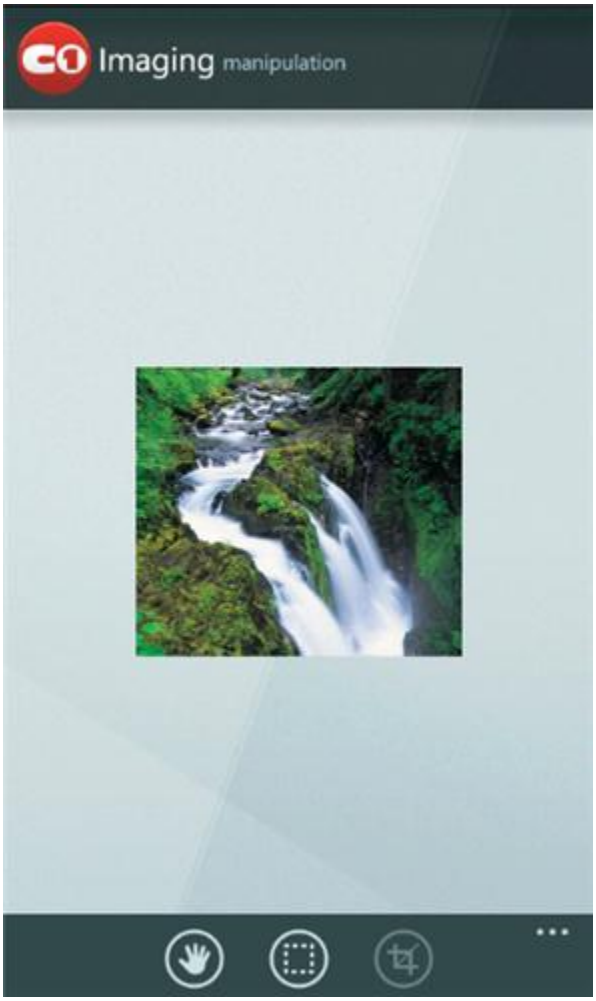
```
// gesture handling
            var gestureListener =
C1.Phone.C1GestureService.GetGestureListener(image);
            gestureListener.Tap += (s, e) =>
            {
                if (manipulationType ==
ImageManipulationGestureType.Select)
                {
                    selection = new Rect(0, 0, bitmap.Width,
bitmap.Height);
                    UpdateMask();
                    btnCrop.IsEnabled = false;
                }
            };
            gestureListener.DragStarted += (s, e) =>
            {
                if (manipulationType ==
ImageManipulationGestureType.Warp)
                {
                    pointOrigin = e.GetPosition(image);
                }
                else if (manipulationType ==
ImageManipulationGestureType.Select)
                {
                    pointOrigin = e.GetPosition(image);
                    pointEnd = e.GetPosition(image);
                    UpdateSelection();
                }
            };
            gestureListener.DragDelta += (s, e) =>
            {
                if (manipulationType ==
ImageManipulationGestureType.Select)
                {
                    pointEnd = e.GetPosition(image);
                    UpdateSelection();
                }
            };
            gestureListener.DragCompleted += (s, e) =>
            {
                if (manipulationType ==
ImageManipulationGestureType.Warp)
                {
                    var pointEnd = e.GetPosition(image);
```

```
                    bitmap = new C1Bitmap(screen);
                    Warp(bitmap, screen, pointOrigin, pointEnd);
                }
                else if (manipulationType ==
ImageManipulationGestureType.Select)
                {
                    btnCrop.IsEnabled = true;
                }
            };
 }
void UpdateSelection()
        {
            var start = pointOrigin;
            var end = pointEnd;

            start.X = Math.Min(Math.Max(start.X, 0), bitmap.Width);
            end.X = Math.Min(Math.Max(end.X, 0), bitmap.Width);
            start.Y = Math.Min(Math.Max(start.Y, 0), bitmap.Height);
            end.Y = Math.Min(Math.Max(end.Y, 0), bitmap.Height);

            selection = new Rect(new Point(
                Math.Round(Math.Min(start.X, end.X)),
                Math.Round(Math.Min(start.Y, end.Y))),
                new Size(Math.Round(Math.Abs(start.X - end.X)),
Math.Round(Math.Abs(start.Y - end.Y))));

            UpdateMask();
        }

        void UpdateMask()
        {
            topMask.Height = selection.Top;
            bottomMask.Height = bitmap.Height - selection.Bottom;
            leftMask.Width = selection.Left;
            rightMask.Width = bitmap.Width - selection.Right;
        }
```



To complete the cropping action, the user simply taps the **Crop** button. The following code performs the crop:

```
        void Crop()
        {
            var crop = new C1Bitmap((int)selection.Width,
(int)selection.Height);
            crop.BeginUpdate();
            for (int x = 0; x < selection.Width; ++x)
            {
                for (int y = 0; y < selection.Height; ++y)
                {
                    crop.SetPixel(x, y, bitmap.GetPixel(x +
(int)selection.X, y + (int)selection.Y));
                }
            }
            crop.EndUpdate();

            LoadImageStream(crop.GetStream(ImageFormat.Png, true));
        }
```



Warping an Image

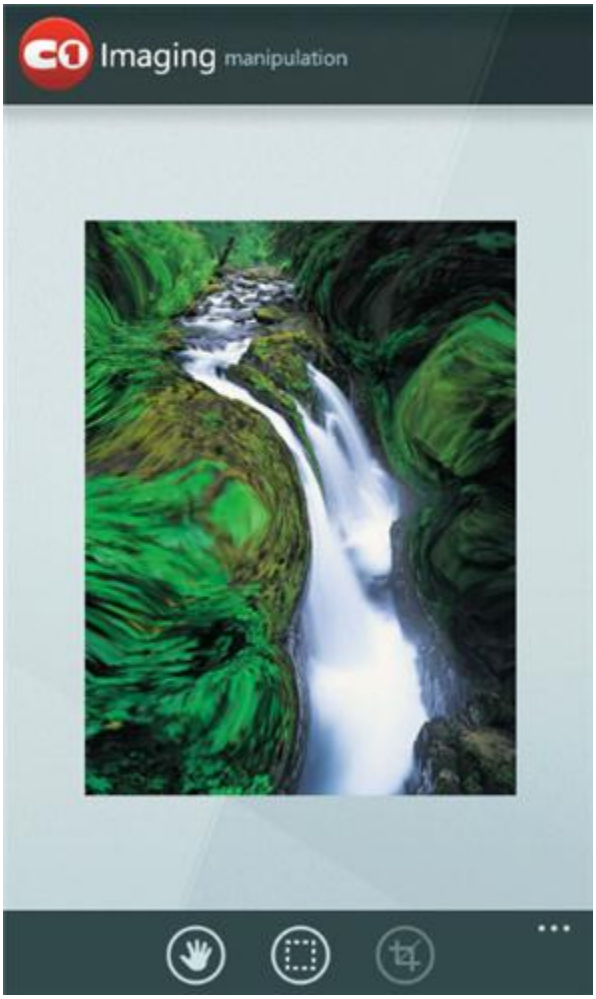
A warping demo for the C1Bitmap component can be seen in the **Imaging manipulation** sample in the **ControlExplorer**. The code uses a lot of math to apply circular transforms throughout the images pixels.

```
void Warp(C1Bitmap src, C1Bitmap dst, Point start, Point end)
        {
            dst.BeginUpdate();
            dst.Copy(src, false);

            var dist = Distance(start, end);
            var affectedDist = dist * 1.5;
            var affectedDistSquared = affectedDist * affectedDist;
            for (int row = 0; row < dst.Height; ++row)
            {
                for (int col = 0; col < dst.Width; ++col)
                {
                    var point = new Point(col, row);
                    if (DistanceSq(start, point) > affectedDistSquared)
                    {
                        continue;
                    }
                    if (DistanceSq(end, point) < 0.25)
                    {
```

```
                        dst.SetPixel(col, row,
src.GetPixel((int)start.X, (int)start.Y));
                        continue;
                    }
                    var dir = new Point(point.X - end.X, point.Y -
end.Y);
                    var t = IntersectRayCircle(end, dir, start,
affectedDist);
                    TryT(-end.X / dir.X, ref t);
                    TryT(-end.Y / dir.Y, ref t);
                    TryT((dst.Width - end.X) / dir.X, ref t);
                    TryT((dst.Height - end.X) / dir.X, ref t);
                    var anchor = new Point(end.X + (point.X - end.X) *
t, end.Y + (point.Y - end.Y));
                    var x = start.X + (anchor.X - start.X) / t;
                    var y = start.Y + (anchor.Y - start.Y) / t;
                    dst.SetPixel(col, row, src.GetInterpolatedPixel(x,
y));
                }
            }
            dst.EndUpdate();
        }

        static double Distance(Point a, Point b)
        {
            return Math.Sqrt(DistanceSq(a, b));
        }

        static double DistanceSq(Point a, Point b)
        {
            var dx = a.X - b.X;
            var dy = a.Y - b.Y;
            return dx * dx + dy * dy;
        }

        static void TryT(double t2, ref double t)
        {
            if (t2 > 0 && t2 < t)
            {
                t = t2;
            }
        }

        static double IntersectRayCircle(Point rayOri, Point rayDir,
Point center, double radius)
        {
            var a = rayDir.X;
            var b = rayOri.X;
            var c = center.X;
            var d = rayDir.Y;
            var e = rayOri.Y;
            var f = center.Y;
            var g = radius * radius;

            var num1 = Math.Sqrt(d * (2 * a * (b - c) * (e - f) - d *
(b * b - 2 * b * c + c * c - g)) - a * a * (e * e - 2 * e * f + f * f -
g));
```

```
            var num2 = a * (c - b) + d * (f - e);
            return (num1 + num2 > 0 ? num1 + num2 : num1 - num2) / (a*a
+ d*d);
        }
```

The user taps the **Warp** button on the application bar menu and touches the screen while dragging to distort the image.



Note: The **ControlExplorer** sample is located in the *C:\Documents and Settings\<username>\My Documents\ComponentOne Samples\Studio for Windows Phone* or *C:\Users\<username>\Documents\ComponentOne Samples\Studio for Windows Phone* folder.

Working with C1GifImage

C1GifImage enables you to add animated GIF files to your Windows Phone applications while providing media player-like commands allowing you to control the GIF animations programmatically.

If you take a look at the **gif** sample for **Imaging for Windows Phone** in the **ControlExplorer**, you'll see an example of an animated .gif.

Note: The **ControlExplorer** sample is located in the *C:\Documents and Settings\<username>\My Documents\ComponentOne Samples\Studio for Windows Phone* or *C:\Users\<username>\Documents\ComponentOne Samples\Studio for Windows Phone* folder.

You'll also notice the pause and play button on the application bar menu.

Adding the animated .gif is as simple as pointing to the location of the image:

```
   _gifImage = new C1GifImage(new
Uri("/ControlExplorer;component/Resources/c1AnimatedGif.gif",
UriKind.Relative));
   image.Source = _gifImage;
```

The following code adds the pause and play buttons for the animated .gif:

```
public partial class GifImages
        : UserControl, IExposeApplicationBarItems
    {
        ApplicationBarMenuItem _btnPlayPause = new
ApplicationBarMenuItem();
        C1GifImage _gifImage;

        public GifImages()
        {
            InitializeComponent();

            _btnPlayPause.Text = "pause";
            _btnPlayPause.Click += new
EventHandler(_btnPlayPause_Click);

            _gifImage = new C1GifImage(new
Uri("/ControlExplorer;component/Resources/c1AnimatedGif.gif",
UriKind.Relative));
            image.Source = _gifImage;
        }
```

```
    void _btnPlayPause_Click(object sender, EventArgs e)
    {
        IsPlaying = !IsPlaying;
    }

    bool _isPlaying = true;
    public bool IsPlaying
    {
        get { return _isPlaying; }
        set
        {
            if (_isPlaying != value)
            {
                _isPlaying = value;
                if (_isPlaying)
                {
                    _gifImage.Play();
                    _btnPlayPause.Text = "pause";
                }
                else
                {
                    _gifImage.Stop();
                    _btnPlayPause.Text = "play";
                }
            }
        }
    }

    public IEnumerable<IApplicationBarMenuItem> ApplicationBarItems
    {
        get
        {
            yield return _btnPlayPause;
        }
    }
}
```