ComponentOne

Imaging for Silverlight

GrapeCity US

GrapeCity
201 South Highland Avenue, Suite 301
Pittsburgh, PA 15206
Tel: 1.800.858.2739 | 412.681.4343
Fax: 412.681.4384
Website: https://www.grapecity.com/en/
E-mail: us.sales@grapecity.com

Trademarks

Warranty

ComponentOne warrants that the media on which the software is delivered is free from defects in material and workmanship, assuming normal use, for a period of 90 days from the date of purchase. If a defect occurs during this time, you may return the defective media to ComponentOne, along with a dated proof of purchase, and ComponentOne will replace it at no charge. After 90 days, you can obtain a replacement for the defective media by sending it and a check for $25 (to cover postage and handling) to ComponentOne.

Except for the express warranty of the original media on which the software is delivered is set forth here, ComponentOne makes no other warranties, express or implied. Every attempt has been made to ensure that the information contained in this manual is correct as of the time it was written. ComponentOne is not responsible for any errors or omissions. ComponentOne's liability is limited to the amount you paid for the product. ComponentOne is not liable for any special, consequential, or other damages for any reason.

Copying and Distribution

Table of Contents

Imaging for Silverlight Library

Load images (PNG and JPG), edit pixel by pixel, and show in an image tag or save to a stream with **Bitmap for Silverlight (C1Bitmap)**.

Display animated GIF images on your Silverlight pages as you would in traditional Web applications with **Image for Silverlight (C1Image)**. Animated GIFs are compact and allow you to add attractive visual elements to your applications with minimal effort.

Help with Silverlight Edition

Getting Started

For information on installing ComponentOne Studio Silverlight Edition, licensing, technical support, namespaces and creating a project with the control, please visit Getting Started with Silverlight Edition.

The C1.Silverlight.Imaging.dll Assembly

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

Main Classes

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

- C1Image**:** Similar to the regular **System.Windows.Media**.**Image** class, but with design-time support for animated GIF images (the regular **Image** element only supports PNG and JPG images).
- 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.

The **C1ImageMagnifier** control was formerly in the **C1.Silverlight.Imaging.dll** assembly. It is now part of **C1.Silverlight.Legacy.dll**. This control is similar to an **Image** element with a 'magnifying glass' over it. The portion of the image under the magnifying glass is zoomed. The user can move the magnifying glass over the image to easily zoom in on any part of the image, and he can use the keyboard to change the zoom factor.

ComponentOne Imaging for Silverlight Library Samples

If you just installed **Silverlight Edition**, open Visual Studio and load the **Samples.sln** solution located in the **Documents\ComponentOne Samples\Silverlight** folder. This solution contains all the samples that ship with this release. Each sample has a readme.txt file that describes it and two projects named as follows:

<SampleName>	Silverlight project (client-side project)
<SampleName>Web	ASP.NET project that hosts the Silverlight project (server-side project)

To run a sample, right-click the **<SampleName>Web** project in the Solution Explorer, select **Set as Startup Project**, and press **F5**.

The following topic describes each of the included samples.

C1.Silverlight.Imaging Samples

The following samples are installed in the **C1.Silverlight.Imaging** folder in the **Samples** directory by default.

Sample Name	Description
C1Imaging Demo	The **C1Imaging_Demo** sample is installed in the **C1.Silverlight.Imaging\C1Imaging_Demo** folder in the samples directory by default. This sample shows samples of the controls in the **C1.Silverlight.Imaging.dll** assembly. This sample consists of the Animated Gif example, which demonstrates how you can load an animated GIF file into a Silverlight application.
Crop	The **Crop** sample is installed in the **C1.Silverlight.Imaging\Crop** folder in the samples directory by default. This sample shows samples of the controls in the **C1.Silverlight.Imaging.dll** assembly. This sample consists of the **Crop2008** sample, which demonstrates how you can load, crop, and export a cropped image file.
FaceWarp	The **FaceWarp** sample is installed in the **C1.Silverlight.Imaging\FaceWarp** folder in the samples directory by default. This sample shows how to use the **C1Bitmap** control to dynamically edit an image. This sample demonstrates **C1Bitmap** by using it to deform images. At run time, drag the mouse over the picture to warp it. You can also load your own image into the sample.

Bitmap

Load images (PNG and JPG), edit pixel by pixel, and show in an image tag or save to a stream with **Bitmap for Silverlight (C1Bitmap)**.

Bitmap for Silverlight Features

The following are some of the main features of **Bitmap for Silverlight** that you may find useful:

- **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.
- **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.
- **Save Generated Images as JPG/PNG**
  With Silverlight 3, you can take screen shots using the **WritableBitmap**, then pass it to C1Bitmap and save the result into a new PNG/JPG file on the fly.

Bitmap for Silverlight Quick Start

The following quick start guide is intended to get you up and running with **Bitmap for Silverlight**. In this quick start, you'll create a new Silverlight application that allows users to load a default image and then crop it. Visual Studio 2010 and Silverlight 4 are used in this example.

Step 1 of 4: Creating a Silverlight Application

In this step you'll create a Silverlight application in Visual Studio using **Bitmap for Silverlight**.

To set up your project, complete the following steps:

1. In Visual Studio, select **File | New | Project**.
2. In the **New Project** dialog box, select a language in the left pane (in this example, C# is used), and in the templates list in the right pane, select **Silverlight Application**.
3. Enter a **Name** for your project and click **OK**. The **New Silverlight Application** dialog box will appear.
4. Click **OK** to accept the default settings. The **MainPage.xaml** file should open.
5. Select **Project | Add Reference**. Browse to find **C1.Silverlight.dll** and **C1.Silverlight.Imaging.dll**, select them, and click **OK**. These .dlls are installed in **C:\Program Files\ComponentOne\Silverlight Edition** by default. They may be installed elsewhere if you installed **Silverlight Edition** to a different location.

In the next step, you'll set the styles and add an image to the project.

Step 2 of 4: Adding an Image

In this step, you will add the following XAML to set the image styles and create a new image.

1. Add the XAML within the *<UserControl>* tags and overwrite the default *<Grid>* tags.

XAML

```
<UserControl.Resources>
        <Style x:Key="CE_SampleText" TargetType="TextBlock">
            <Setter Property="Foreground" Value="#FFF0F8FE" />
            <Setter Property="FontWeight" Value="Normal" />
            <Setter Property="FontSize" Value="11" />
            <Setter Property="HorizontalAlignment" Value="Left"/>
            <Setter Property="VerticalAlignment" Value="Top"/>
            <Setter Property="Width" Value="400"/>
        </Style>
        <Style x:Key="CE_SampleTextBkg" TargetType="Border">
            <Setter Property="Background">
                <Setter.Value>
                    <LinearGradientBrush EndPoint="1,0.5" StartPoint="0,0.5">
                        <GradientStop Color="#99071D2E" Offset="0.003"/>
                        <GradientStop Color="#00071D2E" Offset="1"/>
                    </LinearGradientBrush>
                </Setter.Value>
            </Setter>
            <Setter Property="CornerRadius" Value="2"/>
            <Setter Property="Padding" Value="5 0 0 0"/>
        </Style>
        <SolidColorBrush Color="#55FFFFFF" x:Key="MaskBrush"/>
    </UserControl.Resources>
    <Grid x:Name="LayoutRoot" Background="White">

        <Border BorderBrush="#FF8FB4CC" BorderThickness="3" Grid.Row="2"
VerticalAlignment="Center" HorizontalAlignment="Center">
            <Grid Name="imageGrid">
                <Grid.RowDefinitions>
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="*" />
                    <RowDefinition Height="Auto" />
                </Grid.RowDefinitions>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="Auto" />
                    <ColumnDefinition Width="*" />
                    <ColumnDefinition Width="Auto" />
                </Grid.ColumnDefinitions>
                <Image Stretch="None" Name="image" Grid.RowSpan="3"
Grid.ColumnSpan="3"/>
                <Grid Name="topMask" Grid.ColumnSpan="2"
Background="{StaticResource MaskBrush}" />
                <Grid Name="bottomMask" Grid.Column="1" Grid.Row="2"
Grid.ColumnSpan="2" Background="{StaticResource MaskBrush}" />
                <Grid Name="leftMask" Grid.RowSpan="2" Grid.Row="1"
Background="{StaticResource MaskBrush}" />
```

```
                <Grid Name="rightMask" Grid.Column="2" Grid.RowSpan="2"
Background="{StaticResource MaskBrush}" />
            </Grid>
        </Border>
    </Grid>
```

2. Add an image to the project:
    a. Select **Project | Add Existing Item**.
    b. Browse to find an image. In this example, we use the Lenna.jpg image from the **C1.Silverlight.Imaging\Crop** sample provided with **Silverlight Edition**.
    c. Select the image and click **Add**.

In the next step, you'll add the code used to crop the image.

Step 3 of 4: Adding Code for Image Cropping

The code in this step will load the default image and allow the user to crop it. Follow these steps:

1. Open the **MainPage.xaml.cs** file and add the following **using** (**Imports** in Visual Basic) statements.

C#

```
using C1.Silverlight;
using C1.Silverlight.Imaging;
using System.IO;
```

2. Add the following code to load a default image and define cropping:

C#

```
public partial class MainPage : UserControl
    {

        C1Bitmap bitmap = new C1Bitmap();
        Rect selection;


        public MainPage()
        {
            InitializeComponent();
            LoadDefaultImage();
            image.Source = bitmap.ImageSource;


            var mouseHelper = new C1MouseHelper(imageGrid);
            mouseHelper.MouseDragStart += OnDrag;
            mouseHelper.MouseDragMove += OnDrag;
        }

        void OnDrag(object sender, MouseDragEventArgs e)
        {
```

```
            var transform =
Application.Current.RootVisual.TransformToVisual(image);
            var start = transform.Transform(e.StartPosition);
            var end = transform.Transform(e.CurrentPosition);
            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;
        }

        void LoadDefaultImage()
        {
            LoadImageStream(Application.GetResourceStream(new
Uri("/SilverlightApplication3;component/Lenna.jpg", UriKind.Relative)).Stream);
        }

       void LoadImageStream(Stream stream)
        {
            bitmap.SetStream(stream);

            imageGrid.Width = bitmap.Width;
            imageGrid.Height = bitmap.Height;

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

In the next step you will run the application.

Step 4 of 4: Running the Application

Run the application.

1. From the **Debug** menu, select **Start Debugging** to view the image.
2. Click on the image and keep the left mouse button pressed while dragging the cursor. The **Crop** sample provided with **Silverlight Edition** shows you how to export the cropped image and save it to a file.

In the following image, notice the eye area is cropped.

Congratulations! You have successfully completed the **Image for Silverlight** quick start.

Working with Bitmap for Silverlight

With the official release of Silverlight 4, many familiar features are now possible out of the box. These features include drag and drop management, handling the right mouse button click, and printing. While most of these features were already possible using ComponentOne Silverlight 3 controls, Silverlight 4 adds value and flexibility to our studio as we continue to push the limits of the framework itself.

In this sample we will combine some of the new Silverlight 4 functionality with some classic ComponentOne Silverlight controls to enhance a common type of application; an image editor. Back in the Silverlight 2.0 days, ComponentOne released C1Bitmap, a fully manageable bitmap API for working with images on the client. In this sample we will pull together some different features of **C1Bitmap**, along with new features of Silverlight 4 to build a full image editing application.

The features of Silverlight 4 we will take advantage of are:

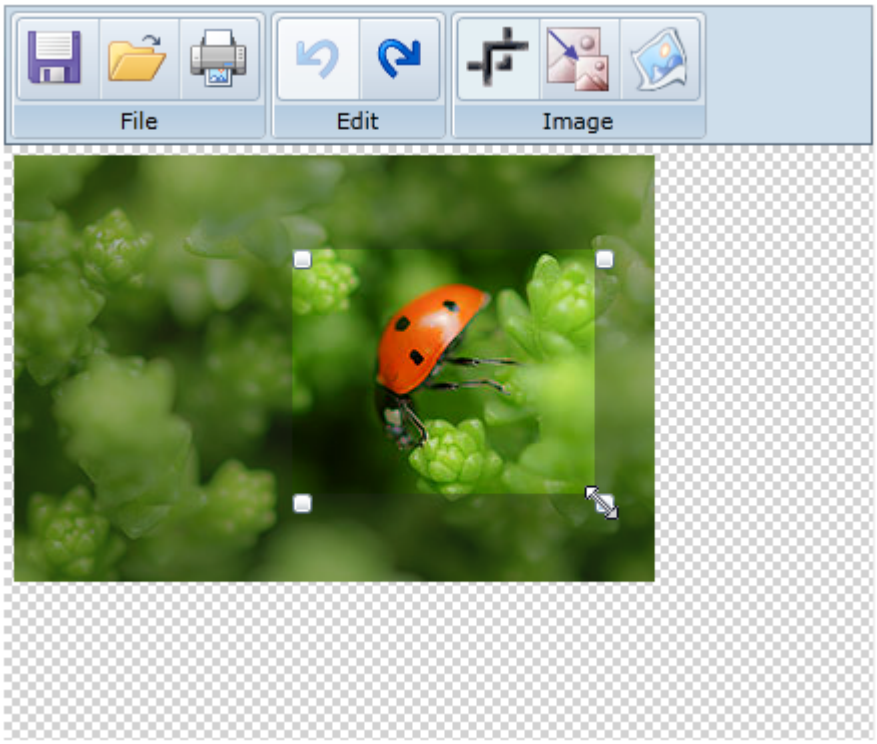
- Drag and Drop onto Silverlight from outside sources
- Printing

Users will be able to drag an image from an outside source, such as their **Documents** folder and drop it into the **Silverlight Image Editor**. Once in the editor, the user will then be able to print the image.

Functionalities we will implement using C1Bitmap are:

- Cropping
- Resizing
- Warping
- Undo/Redo

We will also be using **C1Toolbar** from Studio for Silverlight to complete the application.



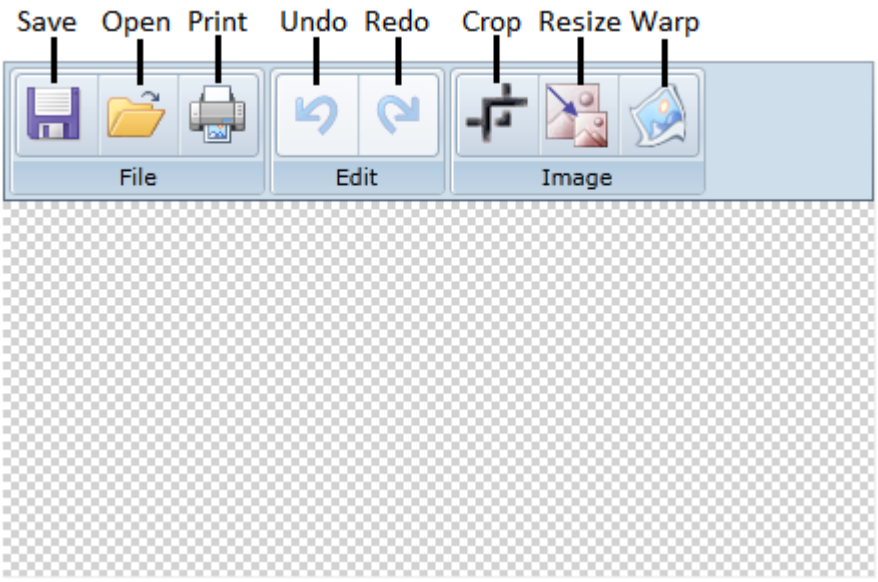
Setting up the Toolbar

Our application will consist of a **C1Toolbar** across the top, and a checkered background filling the remaining region of the application. We are using the **C1Toolbar** with the **C1ToolbarStrip** to create the toolbar, with Ribbon-like groups. **Silverlight Edition** includes a convenient checkered panel control, **C1CheckeredBorder**, which is included in the **C1.Silverlight.Extended** library. This helps with the design of the image editor. Our toolbar consists of 3 **C1ToolbarGroups**: File, Edit and Image.

XAML

```
<c1tb:C1Toolbar Name="c1Toolbar1" Grid.Row="0">
      <c1tb:C1ToolbarGroup Header="File">
          <c1tb:C1ToolbarStrip >
              <c1tb:C1ToolbarButton Name="btnSave" Click="btnSave_Click">
                  <Image Source="Resources/save.png" Margin="2"
ToolTipService.ToolTip="Save"/>
              </c1tb:C1ToolbarButton>
              <c1tb:C1ToolbarButton Name="btnOpen" Click="btnOpen_Click">
                  <Image Source="Resources/Open.png" Margin="2"
ToolTipService.ToolTip="Open"/>
              </c1tb:C1ToolbarButton>
              <c1tb:C1ToolbarButton Name="btnPrint" Click="btnPrint_Click">
                  <Image Source="Resources/Print.png" Margin="2"
ToolTipService.ToolTip="Print"/>
              </c1tb:C1ToolbarButton>
```

```
        </c1tb:C1ToolbarStrip>
    </c1tb:C1ToolbarGroup>
...
</c1tb:C1Toolbar>
```



Adding Drag-and-Drop Behavior

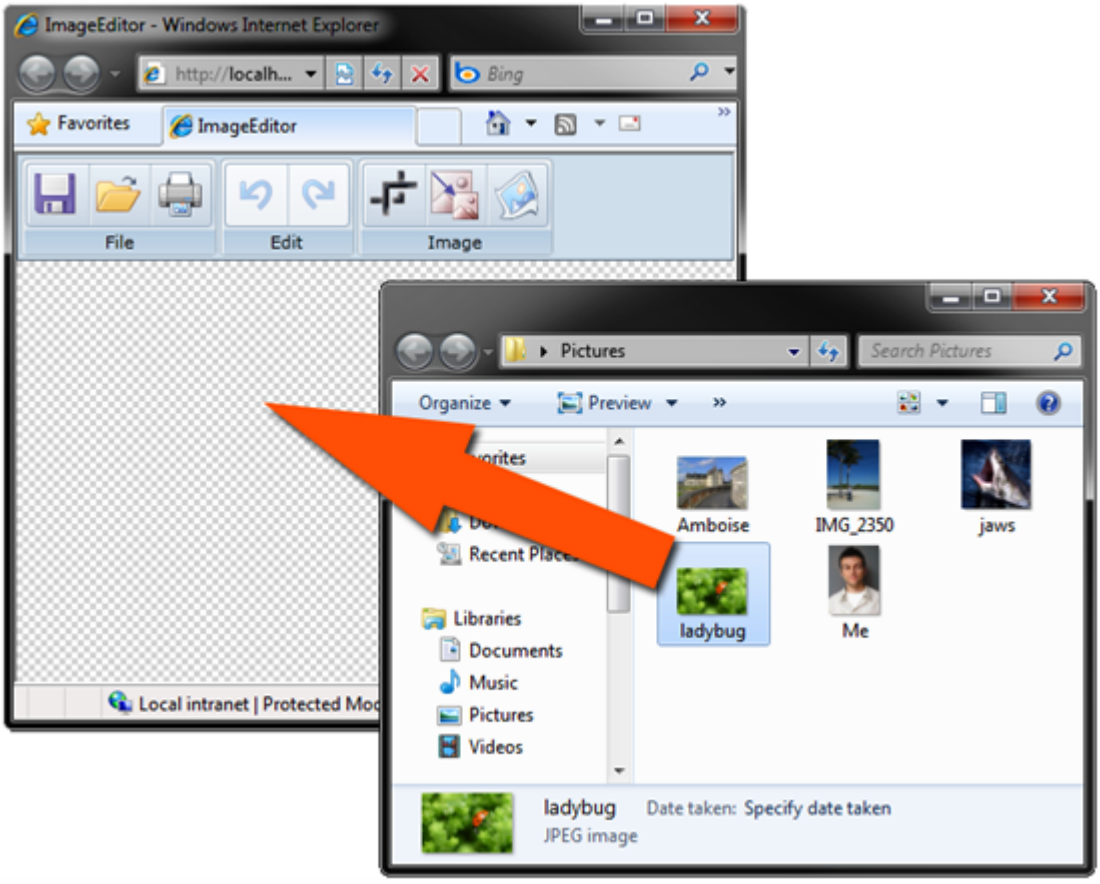
Silverlight 4 gives us inherent drag and drop events on all controls (DragOver, DragLeave, Drop, etc.). This enables us to manage a drag-and-drop process among virtually any elements. But more importantly, Silverlight 4 enables us to drag items from outside the Silverlight application. All you have to do is set the **AllowDrop** property for your container to **True**, and then handle the **Drop** event to grab the files and do what you please. In this sample we will be taking some code from a sample posted by Microsoft. We will use the **C1CheckeredBorder** control as our drop-enabled control.

XAML

```
<c1ext:C1CheckeredBorder Name="checkeredBack" AllowDrop="True" Drop="DropTarget_Drop"
/>
private void DropTarget_Drop(object sender, DragEventArgs e)
 {
     // Get FileInfo array from DragEventArgs
     IDataObject dataObject = e.Data;
     var files = (FileInfo[])dataObject.GetData(DataFormats.FileDrop);
     //Grab first file
     if (files != null)
     {
         FileInfo file = files[0];
         if (IsImageFile(file.Extension))
         {
             Stream stream = file.OpenRead();
```

```
                LoadImageStream(stream);
            }
        }
    }
```

Notice when you drag an image file from your computer onto the Web browser, you see the cursor change to signify a drop. Now, this sample is designed to only accept one file (image extension check is performed), but you can easily drop multiple files at once, as Microsoft's sample demonstrates.



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. We use baked-in **OpenFileDialog** and **SaveFileDialogs** to give our application access to files on the user's machine. 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, resizing and warping.

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. We could, if we wanted, print the entire toolbar with the image too but that's just odd.

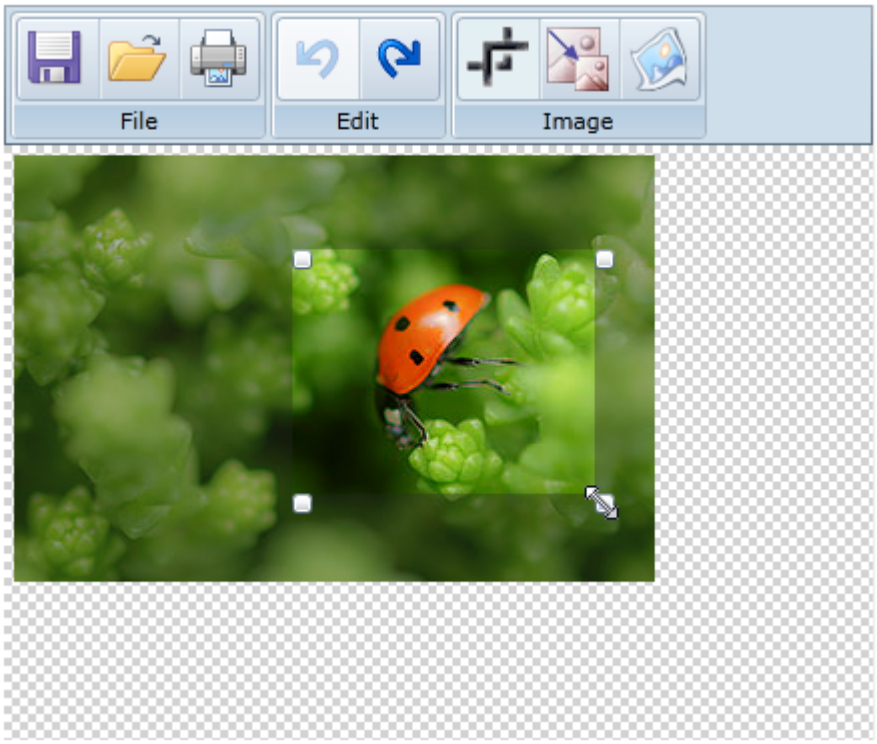
C#

```
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 with a Draggable Crop Box

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. 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.

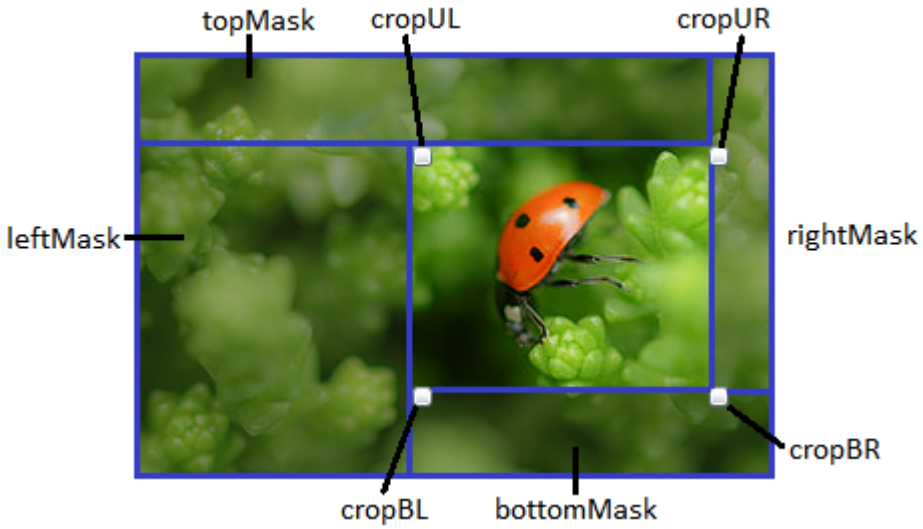
While **C1Bitmap** provides us the API needed to crop the image, it does not however provide us the UI. There are countless ways to implement an image cropping UI. I think most developers and image editors alike prefer to have a draggable box with adorners. This is commonly seen in professional image editing software such as Adobe Photoshop. So that's what we will create.



Here is the XAML that defines the elements needed to create our crop box. It consists of 4 Thumbs which the user can drag, and 4 shaded rectangles which mask the regions that will be cropped out. We place all of these elements in a Canvas so we can easily adjust the positions in code.

XAML

```
<Canvas Name="cropCanvas">
      <Rectangle Name="topMask" Fill="{StaticResource MaskBrush}" Canvas.Top="0"
Canvas.Left="0" />
      <Rectangle Name="bottomMask" Fill="{StaticResource MaskBrush}" />
      <Rectangle Name="leftMask" Fill="{StaticResource MaskBrush}" Canvas.Left="0"/>
      <Rectangle Name="rightMask" Fill="{StaticResource MaskBrush}" Canvas.Top="0" />
      <Thumb Name="cropUL" Width="10" Height="10" DragDelta="cropUL_DragDelta"
Cursor="SizeNWSE" />
      <Thumb Name="cropUR" Width="10" Height="10" DragDelta="cropUR_DragDelta"
Cursor="SizeNESW" />
      <Thumb Name="cropBL" Width="10" Height="10" DragDelta="cropBL_DragDelta"
Cursor="SizeNESW" />
      <Thumb Name="cropBR" Width="10" Height="10" DragDelta="cropBR_DragDelta"
Cursor="SizeNWSE" />
 </Canvas>
```



The code needed to manipulate the crop box is quite complex. It's possible to implement a draggable crop box using behaviors and the Visual State Manager, but a coded solution is definitely easier to understand for novice Silverlight developers. 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 clicking the "Crop" button on the toolbar we will display the crop box at full size. As the user drags the adorners we utilize each Thumb's **DragDelta** event to capture the vertical and horizontal change. Then with a bit of logic and simple math, we can manipulate the behavior of the other adorners which the user is not dragging. To complete the cropping action, the user simply clicks the **Crop** button again (it's a **C1ToolbarToggleButton**).

C#

```
private void cropUL_DragDelta(object sender,
System.Windows.Controls.Primitives.DragDeltaEventArgs e)
 {
     double left = Canvas.GetLeft(cropUL) + e.HorizontalChange;
     double top = Canvas.GetTop(cropUL) + e.VerticalChange;
     if (left > 0 && left < bitmap.Width && cropBox.Width > e.HorizontalChange)
     {
         cropBox = new Rect(left, cropBox.Top, cropBox.Width - e.HorizontalChange,
cropBox.Height);
     }
     if (top > 0 && top < bitmap.Height && cropBox.Height > e.VerticalChange)
     {
         cropBox = new Rect(cropBox.Left, top, cropBox.Width, cropBox.Height -
e.VerticalChange);
     }
     UpdateCropBox();
 }
 private void cropUR_DragDelta(object sender,
System.Windows.Controls.Primitives.DragDeltaEventArgs e)
 {
     double left = Canvas.GetLeft(cropUR) + e.HorizontalChange;
     double top = Canvas.GetTop(cropUR) + e.VerticalChange;

     if (left > 0 && left < bitmap.Width && left > cropBox.Left)
     {
```

```
            cropBox = new Rect(cropBox.Left, cropBox.Top, left - cropBox.Left,
cropBox.Height);
        }
        if (top > 0 && top < bitmap.Height && cropBox.Height > e.VerticalChange)
        {
            cropBox = new Rect(cropBox.Left, top, cropBox.Width, cropBox.Height -
e.VerticalChange);
        }
        UpdateCropBox();
    }
    private void cropBL_DragDelta(object sender,
System.Windows.Controls.Primitives.DragDeltaEventArgs e)
    {
        double left = Canvas.GetLeft(cropBL) + e.HorizontalChange;
        double top = Canvas.GetTop(cropBL) + e.VerticalChange;
        if (left > 0 && left < bitmap.Width && cropBox.Width > e.HorizontalChange)
        {
            cropBox = new Rect(left, cropBox.Top, cropBox.Width - e.HorizontalChange,
cropBox.Height);
        }
        if (top > 0 && top < bitmap.Height && top > cropBox.Top)
        {
            cropBox = new Rect(cropBox.Left, cropBox.Top, cropBox.Width, top -
cropBox.Top);
        }
        UpdateCropBox();
    }
    private void cropBR_DragDelta(object sender,
System.Windows.Controls.Primitives.DragDeltaEventArgs e)
    {
        double left = Canvas.GetLeft(cropBR) + e.HorizontalChange;
        double top = Canvas.GetTop(cropBR) + e.VerticalChange;

        if (left > 0 && left < bitmap.Width && left > cropBox.Left)
        {
            cropBox = new Rect(cropBox.Left, cropBox.Top, left - cropBox.Left,
cropBox.Height);
        }
        if (top > 0 && top < bitmap.Height && cropBox.Height + e.VerticalChange > 0)
        {
            cropBox = new Rect(cropBox.Left, cropBox.Top, cropBox.Width, cropBox.Height
+ e.VerticalChange);
        }
        UpdateCropBox();
    }
```

We apply some logic for the bounds of eachadorner in the "if" statements above. For example, you should not be able to drag the bottom-right adorner further left beyond the bottom-left adorner and so on. And the adorners should not be draggable outside the bounds of the image.

C#

```
private void UpdateCropBox()
 {
     Canvas.SetLeft(cropUL, cropBox.Left);
     Canvas.SetTop(cropUL, cropBox.Top);
     Canvas.SetLeft(cropUR, cropBox.Left + cropBox.Width);
     Canvas.SetTop(cropUR, cropBox.Top);
     Canvas.SetLeft(cropBL, cropBox.Left);
     Canvas.SetTop(cropBL, cropBox.Top + cropBox.Height);
     Canvas.SetLeft(cropBR, cropBox.Left + cropBox.Width);
     Canvas.SetTop(cropBR, cropBox.Top + cropBox.Height);
     UpdateMask();
     cropping = true;
 }
```

The **UpdateCropBox** method updates the position of all the **cropCanvas** elements based upon the **Left**, **Top**, **Width** and **Height** properties of the **cropBox Rect**. When it's time to finally apply the cropping (by clicking the **Crop** button again), **C1Bitmap** joins in on the action as we grab the pixels within the bounding **Rect** and copy them to a new **C1Bitmap**, replacing the original.

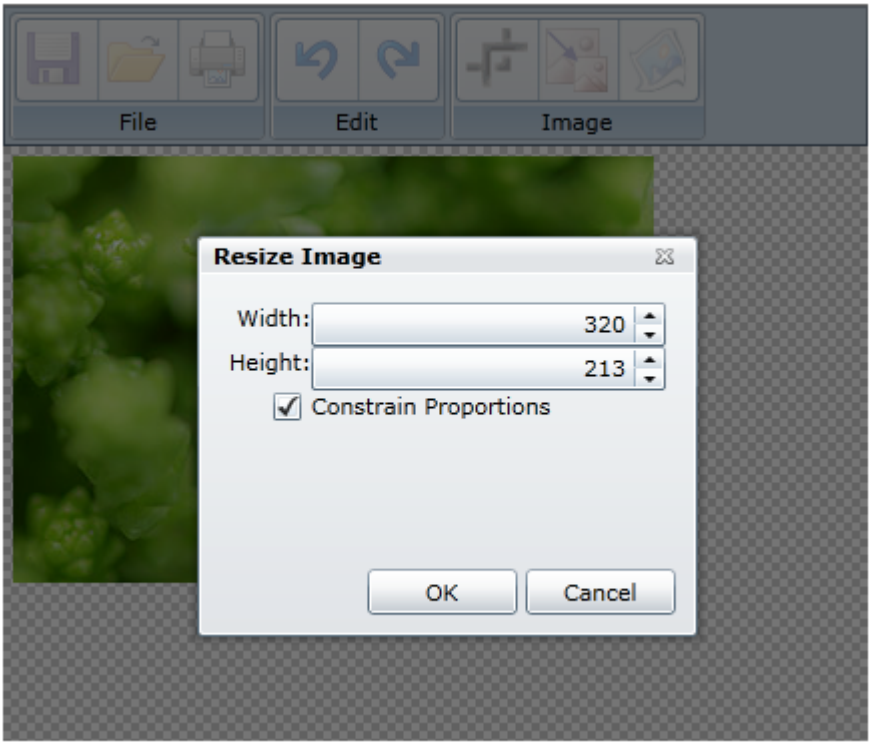
C#

```
private void CropImage()
{
    bitmap2 = new C1Bitmap((int)cropBox.Width, (int)cropBox.Height);
    bitmap2.BeginUpdate();
    for (int x = 0; x < cropBox.Width; ++x)
    {
        for (int y = 0; y < cropBox.Height; ++y)
        {
            bitmap2.SetPixel(x, y, bitmap.GetPixel(x + (int)cropBox.X, y +
(int)cropBox.Y));
        }
    }
    bitmap2.EndUpdate();
    bitmap.Copy(bitmap2, false);
    UpdateImage(true);
    InitCropHandles();
}
```

Resizing the Image

The second most useful image editing task is resizing, or scaling, an image. C1Bitmap provides us with easy ways to resize the image. Most of the work in this part is actually just creating the UI to capture the input from the user. In this sample, we display a **Child Window** prompting for a new **Width** and **Height** for the image. Then we pass these values to our resizing method which will do the work and replace the existing image with the new size. Ideally, we will want to constrain proportions on the resize action but also allow the user to resize freely. This can be handled entirely

through the UI and not rely on the bitmap component.



C#

```
void ResizeImage(int w, int h)
 {
     bitmap = new C1Bitmap(bitmap, w, h);
     UpdateImage(true);
 }
```

Undo and Redo History

We all make mistakes. Having the ability to undo (and then possibly redo) mistakes in any application is extremely convenient because saves us all time from having to start over. In this sample we take a straightforward approach to enabling **Undo/Redo** by saving up to 3 copies of the image after each change is applied. The changes include cropping, resizing and warping. The trick is knowing how to traverse back and forth through the history, while also enabling the user to tack on more additional changes. The code for this really has nothing to do with C1Bitmap or Silverlight 4 enhancements, so it can be used to undo/redo any particular control.

C#

```
List<C1Bitmap> undoBitmaps = new List<C1Bitmap>();
 List<C1Bitmap> redoBitmaps = new List<C1Bitmap>();
 private void btnUndo_Click(object sender, RoutedEventArgs e)
 {
     if (undoBitmaps.Count > 1)
```

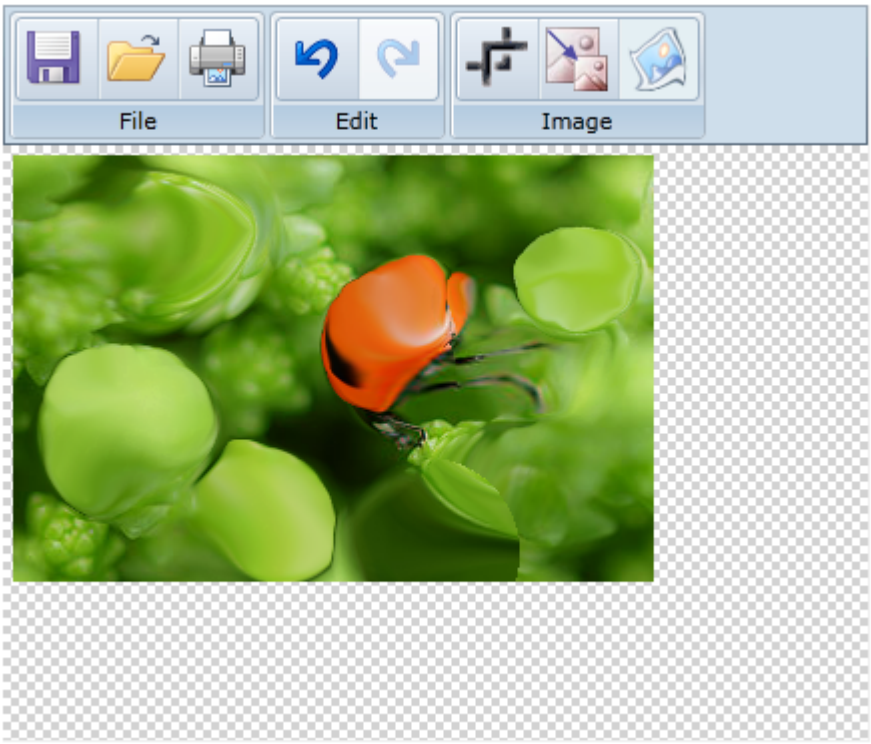
```
    {
        bitmap = new C1Bitmap(undoBitmaps.ElementAt(undoBitmaps.Count - 2));
        redoBitmaps.Add(new C1Bitmap(undoBitmaps.ElementAt(undoBitmaps.Count - 1)));
        undoBitmaps.RemoveAt(undoBitmaps.Count - 1);
        UpdateImage(false);
    }
    UpdateEditButtons();
}
private void btnRedo_Click(object sender, RoutedEventArgs e)
{
    if (redoBitmaps.Count > 0)
    {
        bitmap = new C1Bitmap(redoBitmaps.ElementAt(redoBitmaps.Count - 1));
        undoBitmaps.Add(new C1Bitmap(redoBitmaps.ElementAt(redoBitmaps.Count - 1)));
        redoBitmaps.RemoveAt(redoBitmaps.Count - 1);
        UpdateImage(false);
    }
    UpdateEditButtons();
}

void UpdateHistory()
{
    //Add current bitmap to memory
    undoBitmaps.Add(new C1Bitmap(bitmap));
    redoBitmaps.Clear();

    //Restrict application to only hold up to 3 instances or changes made to
C1Bitmap for undo/redo history
    if (undoBitmaps.Count > 4)
        undoBitmaps.RemoveAt(0);
    UpdateEditButtons();
}
```

Warping - Just for Fun

The initial demo for the C1Bitmap component showed how to manipulate an image pixel by pixel through warping. This can be seen today in the ControlExplorer. The code basically uses a lot of math to apply circular transforms throughout the images pixels. For this sample I have made no changes to this code- I just wanted to add it for fun.


File
Edit
Image

Image

Display animated GIF images on your Silverlight pages as you would in traditional Web applications with **Image for Silverlight**. Animated GIFs are compact and allow you to add attractive visual elements to your applications with minimal effort.

Image for Silverlight Features

The following are some of the main features of **Image for Silverlight** that you may find useful:

- **Support for Animated GIF Files**
  Image enables you to add animated GIF files to your Silverlight applications. The C1Image control can be used to add GIF images at design time (the regular image control only supports PNG and JPEG formats).
- **Play, Pause, and Stop Methods**
  The image source used with the **C1Image** control is the C1GifImage class, which 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.

Image for Silverlight Quick Start

The following quick start guide is intended to get you up and running with **Image for Silverlight**. In this quick start, you'll create a new project in Visual Studio, add a C1Image control to your application, and then run the application. Visual Studio 2010 and Silverlight 4 are used in this example.

Step 1 of 3: Creating a Silverlight Application

In this step you'll create a Silverlight application in Visual Studio using **Image for Silverlight**.

To set up your project and add a C1Image control to your application, complete the following steps:

1. In Visual Studio, select **File | New | Project**.
2. In the **New Project** dialog box, select a language in the left pane (in this example, C# is used), and in the templates list in the right pane, select **Silverlight Application**.
3. Enter a **Name** for your project and click **OK**. The **New Silverlight Application** dialog box will appear.
4. Uncheck the **Host the Silverlight application in a new Web site** box, if necessary, and click **OK**. The **MainPage.xaml** file should open.
5. In the Toolbox, double-click the **C1Image** icon to add the **C1Image** control to **MainPage.xaml**. The XAML markup will now look similar to the following:

XAML

```
<UserControl x:Class="C1Image.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
```

```
    d:DesignHeight="300" d:DesignWidth="400"
xmlns:my="clr-namespace:C1.Silverlight.Imaging;assembly=C1.Silverlight.Imaging">

    <Grid x:Name="LayoutRoot" Background="White">
        <my:C1Image HorizontalAlignment="Left" Margin="170,81,0,0" Name="c1Image1"
VerticalAlignment="Top" />
    </Grid>
</UserControl>
```

Note that the **C1.Silverlight.Imaging** namespace and *<my:C1Image>* tag have been added to the project.

In the next step, you will add an image to the control.

Step 2 of 3: Adding an Image

Next we are going to add an image to the C1Image control.

1. Select the **C1Image** control and in the Visual Studio Properties window, click the **ellipsis** button next to the C1Image.Source property. The **Choose Image** dialog box opens.
2. Click the **Add** button.
3. In the **Open** dialog box, browse to find an image. It can be a .gif (animated or still), .jpg, .jpeg, or .png.
4. Select the image and click **Open**.
5. Click **OK**.

In the next step you will run the application.

Step 3 of 3: Running the Application

Now that you've created a Silverlight application with a C1Image control, you're ready to run the application.

From the **Debug** menu, select **Start Debugging** to view your image.

Congratulations! You have successfully completed the **Image for Silverlight** quick start.

XAML Quick Reference

This topic is dedicated to providing a quick overview of the XAML used to create a C1Image control.

To get started developing, add a **c1** namespace declaration in the root element tag:

```
xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
```

Here is a sample **C1Image** from the **C1Imaging_Demo** sample:

Below is the XAML for the sample:

XAML

```
<UserControlx:Class="C1Imaging_Demo.DemoGifImage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:c1imaging="http://schemas.componentone.com/winfx/2006/xaml"Background="#FF374F5D">
    <Gridx:Name="LayoutRoot"Background="Transparent">
        <PathHeight="200"Width="220"
              Stretch="Fill"
              Data="M 61.3431396484375,0 C61.3431396484375,0 80.2991943359375,38.40919494628906
80.2991943359375,38.40919494628906 80.2991943359375,38.40919494628906
122.686279296875,44.56840515136719 122.686279296875,44.56840515136719
122.686279296875,44.56840515136719 92.01470947265625,74.46580505371094
92.01470947265625,74.46580505371094 92.01470947265625,74.46580505371094
99.25527954101562,116.68159484863281 99.25527954101562,116.68159484863281
99.25527954101562,116.68159484863281 61.3431396484375,96.75 61.3431396484375,96.75
61.3431396484375,96.75 23.430999755859375,116.68159484863281 23.430999755859375,116.68159484863281
23.430999755859375,116.68159484863281 30.67156982421875,74.46580505371094
30.67156982421875,74.46580505371094 30.67156982421875,74.46580505371094 0,44.56840515136719
0,44.56840515136719 0,44.56840515136719 42.3870849609375,38.40919494628906
42.3870849609375,38.40919494628906 42.3870849609375,38.40919494628906 61.3431396484375,0
61.3431396484375,0 z"
              Fill="Black"Stroke="#FF8FB4CC"StrokeThickness="2.5" />

<c1imaging:C1Imagex:Name="image"VerticalAlignment="Center"HorizontalAlignment="Center"Stretch="None"
/>
    </Grid>
</UserControl>
```

Image for Silverlight Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio .NET and know how to use the C1Image control in general. If you are unfamiliar with the **Image for Silverlight** product, please see the Image for Silverlight Quick Start first.

Each topic in this section provides a solution for specific tasks using the **Image for Silverlight** product. Each topic also assumes that you have created a new Silverlight project.

Playing or Stopping an Animated Image

The image source used with the C1Image control is the C1GifImage class, which provides media player-like commands. You can use the C1GifImage.Play, C1GifImage.Stop, and C1GifImage.Pause methods to control GIF animations programmatically. For an example of how to use the **Play** and **Stop** methods, follow these steps:

1. In your Silverlight project, double-click the **C1Image** icon in the Visual Studio Toolbox to add the **C1Image** control to **MainPage.xaml**. The XAML markup will now look similar to the following:

XAML

```
<UserControl x:Class="C1Image.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="400" xmlns:c1imaging="clr-
namespace:C1.Silverlight.Imaging;assembly=C1.Silverlight.Imaging">

    <Grid x:Name="LayoutRoot" Background="White">
        <c1imaging:C1Image HorizontalAlignment="Left" Margin="10,10,0,0"
Name="c1Image1" VerticalAlignment="Top" />
    </Grid>
</UserControl>
```

2. Select the **C1Image** control and in the Properties window, click the **ellipsis** button next to the C1Image.Source property. The **Choose Image** dialog box opens.
3. Click the **Add** button.
4. In the **Open** dialog box, browse to find an animated .gif.
5. Select the image and click **Open**.
6. Click **OK**. You can adjust the size and alignment of the image as necessary.
7. In the Toolbox, double-click the **CheckBox** icon under **Common Silverlight Controls**.
8. In the XAML markup, set the **Content** to **Play**, set the **HorizontalAlignment** to **Center**, and set the **VerticalAlignment** to **Bottom** so your XAML looks similar to the following:

XAML

```
<Grid x:Name="LayoutRoot" Background="White" Height="139" Width="384">
        <c1imaging:C1Image HorizontalAlignment="Center" Margin="10,10,0,252"
Name="c1Image1" Source="Images/Butterfly.gif" Width="44" />
        <CheckBox Content="Play" Height="16" HorizontalAlignment="Center"
Margin="10,10,0,0" Name="checkBox1" VerticalAlignment="Bottom" />
    </Grid>
```

9. Open the MainPage.xaml.cs.
10. Add the following **using** statements (**Imports** if using Visual Basic):
```
using C1.Silverlight.Imaging;
using C1.Silverlight;
```
11. Add code for the **C1GifImage.Play** and **C1GifImage.Stop** methods so it looks similar to the following:

C#

```
public MainPage()
        {
            InitializeComponent();

            var gifImage = new C1GifImage(new Uri("/Images/Butterfly.gif",
UriKind.Relative));
            c1Image1.Source = gifImage;

            checkBox1.IsChecked = true;
            checkBox1.Checked += delegate { gifImage.Play(); };
            checkBox1.Unchecked += delegate { gifImage.Stop(); };
        }
```

12. Click **Debug | Start Debugging** to run the application.
13. Select and clear the **Play** check box to play and stop the animated graphic.