
ComponentOne

Bitmap for WinForms

ComponentOne, a division of GrapeCity

201 South Highland Avenue, Third Floor
Pittsburgh, PA 15206 USA

Website: <http://www.componentone.com>

Sales: sales@componentone.com

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

Trademarks

The ComponentOne product name is a trademark and ComponentOne is a registered trademark of GrapeCity, Inc. All other trademarks used herein are the properties of their respective owners.

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

While you are welcome to make backup copies of the software for your own use and protection, you are not permitted to make copies for the use of anyone else. We put a lot of time and effort into creating this product, and we appreciate your support in seeing that it is used by licensed users only.

Table of Contents

Bitmap for WinForms	2
Help with WinForms Edition	2
Key Features	3
Object Model Summary	4
Quick Start	5-6
Features	7
Loading and Saving an Image	7-8
Applying Transformations	8-9
Clipping an Image	9-12
Flipping an Image	12-14
Rotating an Image	14-15
Scaling an Image	15-17
Working with Bitmap	18
Applying Direct2D Effects	18-23
Bitmap Samples	24

Bitmap for WinForms

ComponentOne Studio introduces **Bitmap for WinForms**, a class library designed to load, save, and transform images. Using Bitmap, you can clip, flip, scale, rotate, or apply any arbitrary combination of these transformations on an image file. In addition, Bitmap allows you to change the pixel format of an image, and supports various container formats such as BMP, PNG, JPG, etc. to cater diverse image processing needs.



Help with WinForms Edition

For information on installing **ComponentOne Studio WinForms Edition**, licensing, technical support, namespaces, and creating a project with the controls, please visit [Getting Started with WinForms Edition](#).

Key Features

Bitmap offers many advanced image processing features beyond simple image loading and saving as listed below:

- **Load images**

Bitmap loads images of various container formats including BMP, PNG, JPEG, JPEG-XR, and ICO. Bitmap also supports single frame TIFF and GIFs. In addition, Bitmap allows loading several images, one by one, into the same instance of C1Bitmap.

- **Save images**

As with loading, the image loaded in a Bitmap can be saved into a storage file, memory stream, or another bitmap object. In addition, Bitmap provides separate SaveAs methods for each of the supported container formats.



Bitmap does not support saving an image in ICO format.

- **Transform images**

With Bitmap, you can apply various transformations on an image. For instance, you can easily clip, crop, rotate, scale in and scale out an image by applying transformation.

- **Apply Direct2D effects**

Bitmap allows you to create varied animations and imaging effects by applying Direct2D effects on an image.

Object Model Summary

Bitmap comes with a rich object model, providing various classes, objects, collections, and associated methods and properties for processing images. The following table lists some of these objects and their major properties.

C1Bitmap
Properties: HasImage , HasMetadata , ImagingFactory , IsDisposed , NativeBitmap , PixelFormat , PixelHeight , PixelWidth
Methods: Import , Load , Save , Transform
Clipper
Property: ImageRect
FlipRotator
Property: TransformOptions
FormatConverter
Properties: DestinationFormat , Palette , PaletteTranslate
Scaler
Properties: DestinationHeight , DestinationWidth , InterpolationMode

Quick Start

This quick start section gets you started with using **Bitmap** for loading an image. You begin by creating a WinForms application in Visual Studio, adding a sample image to your application, and adding code to load the sample image in a picture box using Bitmap. The code given in this section illustrates loading an image into bitmap through a stream object.

Complete the steps given below to see how Bitmap can be used to load an image in a picture box.

1. **Setting up the application and adding a sample image**
2. **Adding code to load image using Bitmap**

The following image shows how the application displays an image loaded in bitmap on a button click.



Step 1: Setting up the application and adding a sample image

1. Create a **WinForms** application in Visual Studio.
2. Add the following references to your application.
 - o C1.Win.4
 - o C1.Win.Bitmap.4
 - o C1.Win.C1DX.4
3. In the **Solution Explorer**, right click your project name and select **Add | New Folder** and name it as '**Resources**'.
4. In Visual Studio, add a sample image to the Resources folder and set its **Build Action** property to **Embedded Resource** from the Properties pane.
5. Add a standard **Button** control for loading a sample image on button click, and a **PictureBox** control for displaying the sample image onto the Form.
6. Set the **Text** property of the button to a suitable text from the Properties pane.
7. Set the **SizeMode** property of the picture box to **StretchImage** from the Properties pane.

Step 2: Adding code to load image using Bitmap

1. Switch to the code view and add the following import statements.

```
Imports Visual Basic
Imports C1.Win.Bitmap
Imports C1.Util.DX
Imports System.Reflection
```


- ```
Imports System.IO
 o C#
using Cl.Win.Bitmap;
using Cl.Util.DX;
using System.Reflection;
using System.IO;
```
- Initialize a bitmap in the Form1 class.
    - Visual Basic**

```
'Initialize a Bitmap
Dim bitmap As New ClBitmap()
```
    - C#**

```
//Initialize a Bitmap
ClBitmap bitmap = new ClBitmap();
```
  - Subscribe a button click event and add the following code for loading the sample image into bitmap from a stream object.
    - Visual Basic**

```
'Load image through stream on button click
Private Sub Btn_Load_Click(sender As Object, e As EventArgs) Handles Btn_Load.Click
 Dim t As Type = Me.GetType
 Dim asm As Assembly = t.Assembly
 Dim stream As Stream = asm.GetManifestResourceStream(t, "GrapeCity.png")
 bitmap.Load(stream, New FormatConverter(PixelFormat.Format32bppPBGRA))
 UpdateImage()
End Sub
```
    - C#**

```
//Load image through stream on button click
private void button1_Click(object sender, EventArgs e)
{
 Assembly asm = typeof(Form1).Assembly;
 using (Stream stream = asm.GetManifestResourceStream(
 "LoadBitmapStream.Resources.GrapeCity.png"))
 {
 bitmap.Load(stream,
 new FormatConverter(PixelFormat.Format32bppPBGRA));
 }

 UpdateImage();
}
```
  - Add the following code to define UpdateImage method for displaying the image in the picture box.
    - Visual Basic**

```
'Display the image loaded in bitmap
Private Sub UpdateImage()
 Dim bmp = pictureBox1.Image
 bmp = bitmap.ToGdiBitmap()
 pictureBox1.Image = bmp
 pictureBox1.Width = bmp.Width
 pictureBox1.Height = bmp.Height

End Sub
```
    - C#**

```
//Display the image loaded in bitmap
private void UpdateImage()
{
 var bmp = pictureBox1.Image as Bitmap;
 bmp = bitmap.ToGdiBitmap();
 pictureBox1.Image = bmp;
}
```



## Features

Bitmap supports a number of features to help users process and handle images.

### [Loading and saving an image](#)

Learn how to implement loading and saving in code.

### [Applying transformations](#)

Learn how to apply different transformations in code.

## Loading and Saving an Image

Bitmap comes with various methods to load images. The [C1Bitmap](#) class provides several [Load](#) method overloads to load image from various sources such as a file or memory stream. It also allows you to load image metadata, which can be used to determine image size, pixel format, or resolution (in dots-per-inch).

The loaded image can be saved to a file or a memory stream. The C1Bitmap class provides general [Save](#) methods that accept the container format as an argument. C1Bitmap also provides separate **SaveAs** methods for each of the supported container formats.

The following code illustrates loading and saving an arbitrary image on button clicks. The code example uses **OpenFileDialog** and **SaveFileDialog** to access an image file kept anywhere on the user's machine. To know how an image can be loaded from a stream object, see the [Quick start](#) section.

- **Visual Basic**

```
Partial Public Class Form1
 Inherits Form
 'Defining a global variable for bitmap
 Private bitmap As C1Bitmap

 Public Sub New()
 InitializeComponent()

 'Initializing a bitmap
 bitmap = New C1Bitmap()
 End Sub

 'Event to load an arbitrary image into picture box on button click
 Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
 Dim ofd = New OpenFileDialog()
 ofd.Filter = "Image Files|*.ico;*.bmp;" +
 "*.gif;*.png;*.jpg;*.jpeg;*.jxr;*.tif;*.tiff"
 ofd.Title = "Select the Image"

 If ofd.ShowDialog() = DialogResult.OK Then
 bitmap.Load(ofd.FileName,
 New FormatConverter(PixelFormat.Format32bppPBGRA))
 PictureBox1.Image = bitmap.ToGdiBitmap()
 End If
 End Sub

 'Event to save the image appearing in picture box to a file on button click
 Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
 Dim sfd = New SaveFileDialog()
 sfd.Filter = "Png Files (*.png)|*.png"
 sfd.CheckPathExists = True
 If sfd.ShowDialog() = DialogResult.OK Then
 bitmap.Save(sfd.FileName, ContainerFormat.Png)
 End If
 End Sub
End Class
```

```
 End If
 End Sub
End Class
```

- **C#**

```
public partial class Form1 : Form
{
 //Defining a global variable for bitmap
 C1Bitmap bitmap;

 public Form1()
 {
 InitializeComponent();

 //Initializing a bitmap
 bitmap = new C1Bitmap();
 }

 //Event to load an arbitrary image into picture box on button click
 private void button1_Click(object sender, EventArgs e)
 {
 var ofd = new OpenFileDialog();
 ofd.Filter = "Image Files|*.ico;*.bmp;" +
 "*.gif;*.png;*.jpg;*.jpeg;*.jxr;*.tif;*.tiff";
 ofd.Title = "Select the Image";

 if (ofd.ShowDialog() == DialogResult.OK)
 {
 bitmap.Load(ofd.FileName,
 new FormatConverter(PixelFormat.Format32bppPBGRA));
 pictureBox1.Image = bitmap.ToGdiBitmap();
 }
 }

 //Event to save the image appearing in picture box to a file on button click
 private void button2_Click(object sender, EventArgs e)
 {
 var sfd = new SaveFileDialog();
 sfd.Filter = "Png Files (*.png)|*.png";
 sfd.CheckPathExists = true;
 if (sfd.ShowDialog() == DialogResult.OK)
 {
 bitmap.Save(sfd.FileName, ContainerFormat.Png);
 }
 }
}
```

## Applying Transformations

Bitmap allows you to apply various transformations on images, such as clipping, flipping, scaling, and rotating. Learn about these transformations and how they can be implemented.

### [Clipping an image](#)

Learn how to implement clipping in code.

### [Flipping an image](#)

Learn how to implement flipping in code.

### [Rotating an image](#)

Learn how to implement rotating in code.

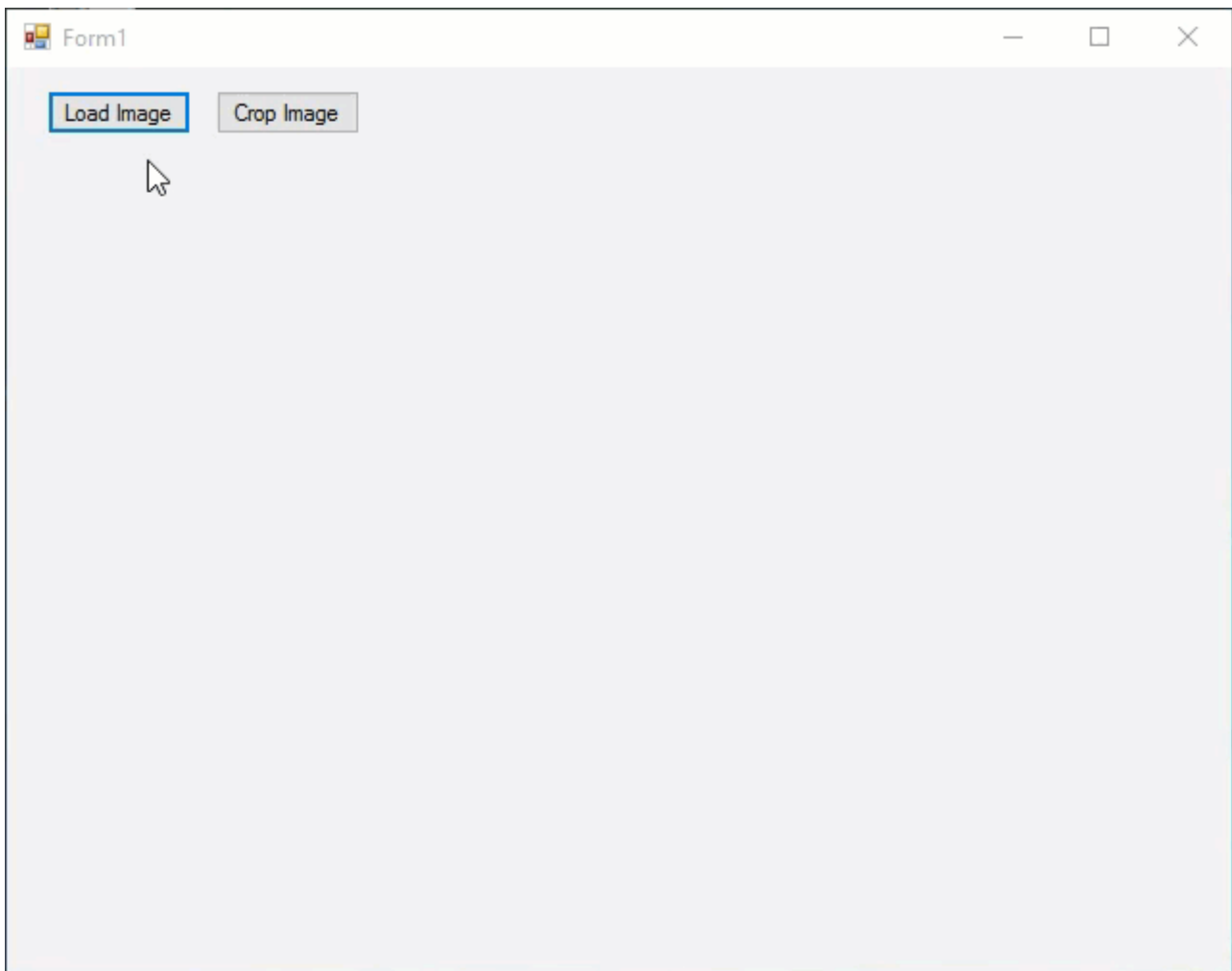
[Scaling an image](#)

Learn how to implement scaling in code.

## Clipping an Image

In two-dimensional images, clipping is an essential requirement as it provides selective rendering of pixels within the boundaries of the selected frame area. Bitmap lets you clip the source image and load a part of the whole image using Clipper transformation.

The following image shows the clipping feature.



Complete the following steps to clip an image using Bitmap in code.

1. Add the following import statement.

- **Visual Basic**

```
Imports System.Drawing.Drawing2D
```

- **C#**

```
using System.Drawing.Drawing2D;
```

2. Initialize a rectangle and a point as global variables in Form1 class.

- **Visual Basic**

```
'Initialize a rectangle and a point
```

```
Dim selection As New RectF(1.0F, 1.0F)
```

```
Dim start As Point2L
```

- **C#**

```
//Initialize a rectangle and a point
RectF selection = new RectF(1f, 1f);
Point2L start;
```

3. Add the following code to apply clipper transformation.

- o **Visual Basic**

```
'Transform method to apply transformation
Private Sub ApplyTransform(t As BaseTransform)
 Dim newBitmap = bitmap.Transform(t)
 bitmap.Dispose()
 bitmap = newBitmap
 selection = New RectF(1.0F, 1.0F)
 UpdateImage()
End Sub

'Event to apply clipper transformation on button click
Private Sub Btn_Clip_Click(sender As Object, e As EventArgs) _
 Handles Btn_Clip.Click
 Dim rect = New RectF(selection.X * bitmap.PixelWidth,
 selection.Y * bitmap.PixelHeight,
 selection.Width * bitmap.PixelWidth,
 selection.Height * bitmap.PixelHeight)
 ApplyTransform(New Clipper(New ImageRect(rect.Round()))))
End Sub
```

- o **C#**

```
//Transform method to apply transformation
void ApplyTransform(BaseTransform t)
{
 var newBitmap = bitmap.Transform(t);
 bitmap.Dispose();

 bitmap = newBitmap;
 selection = new RectF(1f, 1f);

 UpdateImage();
}

//Event to apply clipper transformation on button click
private void button2_Click(object sender, EventArgs e)
{
 var rect = new RectF(selection.X * bitmap.PixelWidth,
 selection.Y * bitmap.PixelHeight,
 selection.Width * bitmap.PixelWidth,
 selection.Height * bitmap.PixelHeight);

 ApplyTransform(new Clipper(new ImageRect(rect.Round())));
}
```

4. Add the following code to select a portion of the image to be clipped.

- o **Visual Basic**

```
'Events to select a portion of image using mouse
Private Sub pictureBox1_MouseClick(sender As Object, e As MouseEventArgs) _
 Handles pictureBox1.MouseClick
 If (e.Button And MouseButton.Left) <> 0 Then
 Dim dcs = SystemInformation.DoubleClickSize
 If Math.Abs(e.X - start.X) _
 < dcs.Width AndAlso Math.Abs(e.Y - start.Y) < dcs.Height Then
 selection = New RectF(1.0F, 1.0F)
 pictureBox1.Invalidate()
 End If
 End If
End Sub
```

```

Private Sub pictureBox1_MouseDown(sender As Object, e As MouseEventArgs) _
 Handles pictureBox1.MouseDown
 If (e.Button And MouseButton.Left) <> 0 Then
 start = New Point2L(e.X, e.Y)
 End If
End Sub

Private Sub pictureBox1_MouseMove(sender As Object, e As MouseEventArgs) _
 Handles pictureBox1.MouseMove
 If (e.Button And MouseButton.Left) <> 0 Then
 Dim w As Integer = pictureBox1.Width
 Dim h As Integer = pictureBox1.Height
 Dim x As Integer = Math.Max(0, Math.Min(e.X, w))
 Dim y As Integer = Math.Max(0, Math.Min(e.Y, h))

 selection = New RectF(CSng(Math.Min(start.X, x)) / w,
 CSng(Math.Min(start.Y, y)) / h,
 CSng(Math.Abs(x - start.X)) / w,
 CSng(Math.Abs(y - start.Y)) / h)

 pictureBox1.Invalidate()
 End If
End Sub

Private Sub pictureBox1_Paint(sender As Object, e As PaintEventArgs) _
 Handles pictureBox1.Paint
 Dim w As Integer = pictureBox1.Width
 Dim h As Integer = pictureBox1.Height
 Dim path = New GraphicsPath(FillMode.Alternate)
 path.AddRectangle(New RectangleF(0, 0, w, h))
 path.AddRectangle(New RectangleF(selection.X * w,
 selection.Y * h,
 selection.Width * w,
 selection.Height * h))

 Dim brush = New SolidBrush(Color.FromArgb(&H66FFFFFF))
 e.Graphics.FillPath(brush, path)
 brush.Dispose()

 path.Dispose()
End Sub

```

o **C#**

```

//Events to select a portion of image from the picture box using mouse
private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
{
 if ((e.Button & MouseButton.Left) != 0)
 {
 start = new Point2L(e.X, e.Y);
 }
}

private void pictureBox1_MouseClick(object sender, MouseEventArgs e)
{
 if ((e.Button & MouseButton.Left) != 0)
 {
 var dcs = SystemInformation.DoubleClickSize;
 if (Math.Abs(e.X - start.X)
 < dcs.Width && Math.Abs(e.Y - start.Y) < dcs.Height)
 {
 selection = new RectF(1f, 1f);
 pictureBox1.Invalidate();
 }
 }
}

```

```
 }
}

private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
{
 if ((e.Button & MouseButtons.Left) != 0)
 {
 int w = pictureBox1.Width;
 int h = pictureBox1.Height;
 int x = Math.Max(0, Math.Min(e.X, w));
 int y = Math.Max(0, Math.Min(e.Y, h));

 selection = new RectF(
 (float)Math.Min(start.X, x) / w,
 (float)Math.Min(start.Y, y) / h,
 (float)Math.Abs(x - start.X) / w,
 (float)Math.Abs(y - start.Y) / h);

 pictureBox1.Invalidate();
 }
}

private void pictureBox1_Paint(object sender, PaintEventArgs e)
{
 int w = pictureBox1.Width;
 int h = pictureBox1.Height;
 var path = new GraphicsPath(FillMode.Alternate);
 path.AddRectangle(new RectangleF(0, 0, w, h));
 path.AddRectangle(new RectangleF(selection.X * w, selection.Y * h,
 selection.Width * w, selection.Height * h));

 var brush = new SolidBrush(Color.FromArgb(0x66FFFFFF));
 e.Graphics.FillPath(brush, path);
 brush.Dispose();

 path.Dispose();
}
```

5. Press F5 to run the application and click 'Load Image' button to load an image.
6. Select a portion of the image through mouse and click 'Clip Image' button to crop the selected portion.

## Flipping an Image

Bitmap lets you flip an image vertically or horizontally. To produce a flipped image using Bitmap, you can set the [TransformOptions](#) property of the [FlipRotator](#) class. The TransformOption property can be set through [TransformOptions](#) enumeration.

The image below shows a horizontally-flipped image.



The following code illustrates flipping an image vertically or horizontally on button clicks. This example uses the sample created in the [Quick start](#) section.

- **Visual Basic**

```
Private Sub ApplyTransform(t As BaseTransform)
 Dim newBitmap = bitmap.Transform(t)
 bitmap.Dispose()
 bitmap = newBitmap
 selection = New RectF(1.0F, 1.0F)
 UpdateImage()
End Sub

'Event to flip the image vertically on button click
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
 ApplyTransform(New FlipRotator(TransformOptions.FlipVertical))
End Sub

'Event to flip the image horizontally on button click
Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
 ApplyTransform(New FlipRotator(TransformOptions.FlipHorizontal))
End Sub
```

- **C#**

```
void ApplyTransform(BaseTransform t)
{
 var newBitmap = bitmap.Transform(t);
 bitmap.Dispose();

 bitmap = newBitmap;
 selection = new RectF(1f, 1f);
}
```



```
UpdateImage();
}

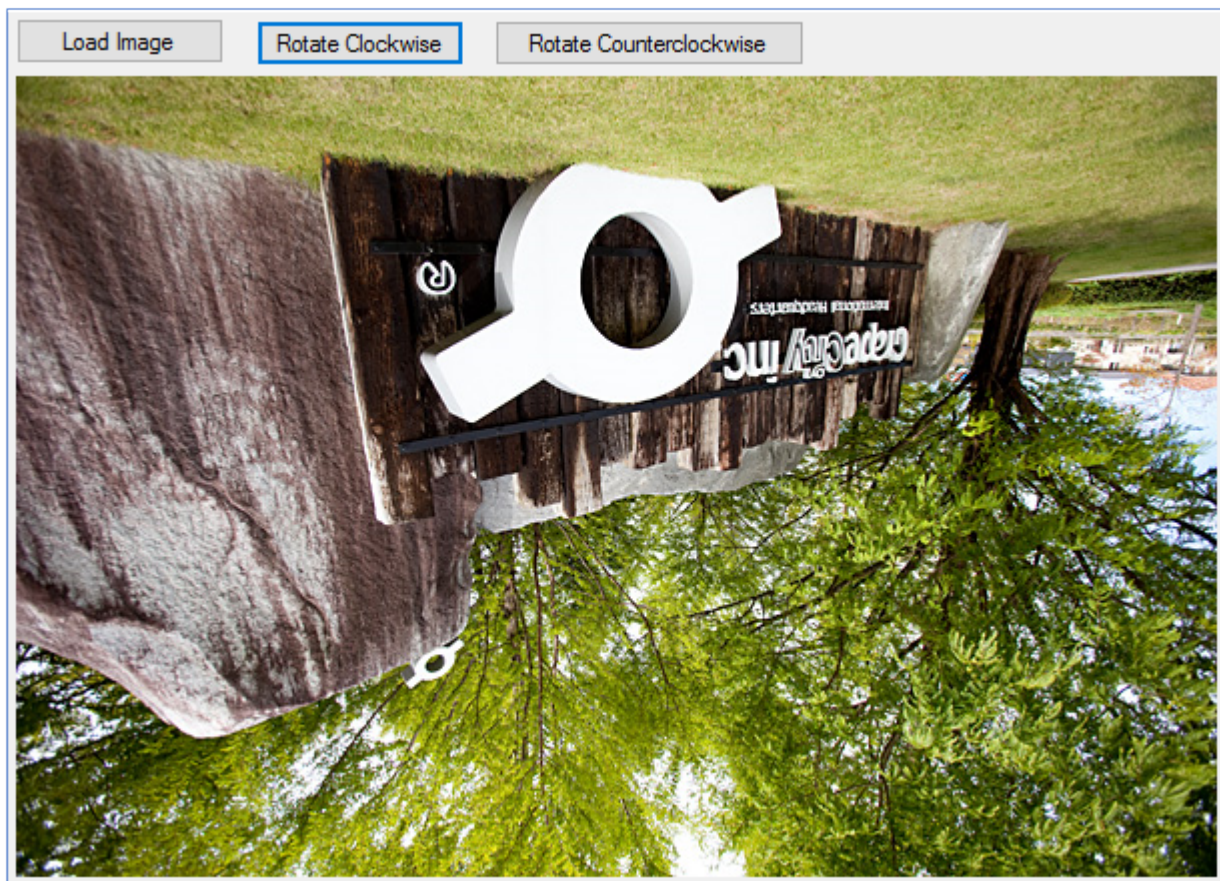
//Event to flip the image vertically on button click
private void button3_Click(object sender, EventArgs e)
{
 ApplyTransform(new FlipRotator(TransformOptions.FlipVertical));
}

//Event to flip the image horizontally on button click
private void button4_Click(object sender, EventArgs e)
{
 ApplyTransform(new FlipRotator(TransformOptions.FlipHorizontal));
}
```

## Rotating an Image

Bitmap lets you rotate an image to 90 degree, 180 degree and 270 degree in clockwise direction. To rotate an image using Bitmap, you can set the [TransformOptions](#) property of the [FlipRotator](#) class. The TransformOption property can be set through the [TransformOptions](#) enumeration.

The image below shows an image rotated by 180 degree clockwise.



The following code illustrates rotating an image in clockwise and counterclockwise directions on button clicks. This example uses the sample created in the [Quick start](#) section.

- **Visual Basic**

```
Private Sub ApplyTransform(t As BaseTransform)
```

```
Dim newBitmap = bitmap.Transform(t)
bitmap.Dispose()
bitmap = newBitmap
selection = New RectF(1.0F, 1.0F)
UpdateImage()
End Sub

'Event to rotate the image in clockwise direction on button click
Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
 ApplyTransform(New FlipRotator(TransformOptions.Rotate180))
End Sub

'Event to rotate the image in counterclockwise direction on button click
Private Sub Button3_Click(sender As Object, e As EventArgs) Handles Button3.Click
 ApplyTransform(New FlipRotator(TransformOptions.Rotate270))
End Sub
```

- **C#**

```
void ApplyTransform(BaseTransform t)
{
 var newBitmap = bitmap.Transform(t);
 bitmap.Dispose();

 bitmap = newBitmap;
 selection = new RectF(1f, 1f);

 UpdateImage();
}

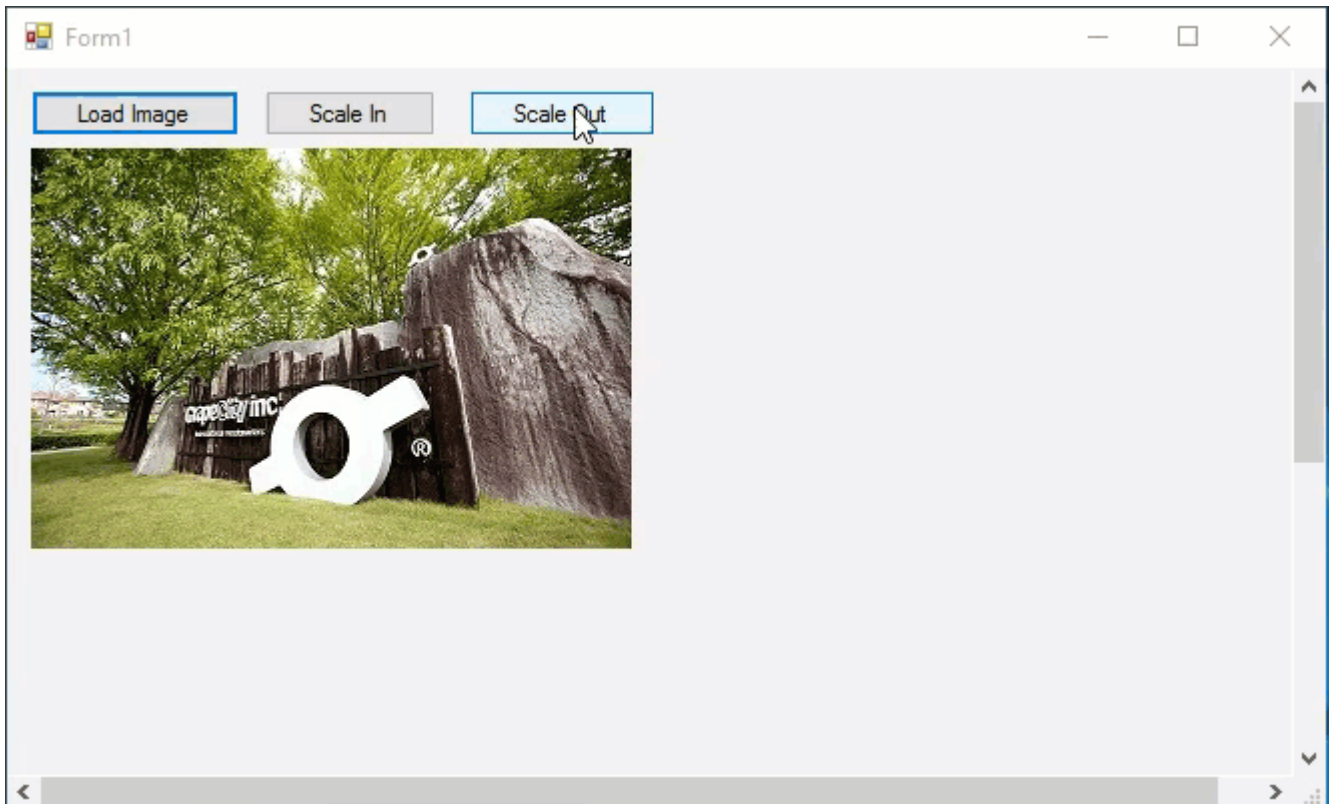
//Event to rotate the image in clockwise direction on button click
private void button3_Click(object sender, EventArgs e)
{
 ApplyTransform(new FlipRotator(TransformOptions.Rotate180));
}

//Event to rotate the image in counterclockwise direction on button click
private void button4_Click(object sender, EventArgs e)
{
 ApplyTransform(new FlipRotator(TransformOptions.Rotate270));
}
```

## Scaling an Image

Scaling is an important requirement of image processing as it resizes (increases and decreases the size) image. Bitmap also allows scaling in and out an image through the [InterpolationMode](#) property of the [Scaler](#) class.

The image below shows scaling in and scaling out feature.



The following code illustrates scaling in and scaling out an image on button clicks. This example uses the sample created in the [Quick start](#) section.

- **Visual Basic**

```
Private Sub ApplyTransform(t As BaseTransform)
 Dim newBitmap = bitmap.Transform(t)
 bitmap.Dispose()
 bitmap = newBitmap
 selection = New RectF(1.0F, 1.0F)
 UpdateImage()
End Sub

'Event to scale out the image on button click
Private Sub Button2_Click(sender As Object, e As EventArgs) _
 Handles Button2.Click
 Dim px As Integer = CInt(bitmap.PixelWidth * 1.6F + 0.5F)
 Dim py As Integer = CInt(bitmap.PixelHeight * 1.6F + 0.5F)
 ApplyTransform(New Scaler(px, py, _
 C1.Win.Bitmap.InterpolationMode.HighQualityCubic))
End Sub

'Event to scale in the image on button click
Private Sub Button3_Click(sender As Object, e As EventArgs) _
 Handles Button3.Click
 Dim px As Integer = CInt(bitmap.PixelWidth * 0.625F + 0.5F)
 Dim py As Integer = CInt(bitmap.PixelHeight * 0.625F + 0.5F)
 If px > 0 AndAlso py > 0 Then
 ApplyTransform(New Scaler(px, py, _
 C1.Win.Bitmap.InterpolationMode.HighQualityCubic))
 End If
End Sub
```

- **C#**

```
void ApplyTransform(BaseTransform t)
{
 var newBitmap = bitmap.Transform(t);
 bitmap.Dispose();

 bitmap = newBitmap;
 selection = new RectF(1f, 1f);

 UpdateImage();
}

//Event to scale out the image on button click
private void button3_Click(object sender, EventArgs e)
{
 int px = (int)(bitmap.PixelWidth * 1.6f + 0.5f);
 int py = (int)(bitmap.PixelHeight * 1.6f + 0.5f);
 ApplyTransform(new Scaler(px, py,
 C1.Win.Bitmap.InterpolationMode.HighQualityCubic));
}

//Event to scale in the image on button click
private void button4_Click(object sender, EventArgs e)
{
 int px = (int)(bitmap.PixelWidth * 0.625f + 0.5f);
 int py = (int)(bitmap.PixelHeight * 0.625f + 0.5f);
 if (px > 0 && py > 0)
 {
 ApplyTransform(new Scaler(px, py,
 C1.Win.Bitmap.InterpolationMode.HighQualityCubic));
 }
}
```



## Working with Bitmap

Working with Bitmap section assumes that you are familiar with the basic features of the Bitmap control and know how to use it in general. The following section provides information on auxiliary functionality offered by Bitmap.

### Applying Direct2DEffects

Learn how to apply Direct2D effects in code.

## Applying Direct2D Effects

**Direct2D** is a two-dimensional graphics API designed by Microsoft that offers a range of built-in and custom effects for manipulating images. The API provides high quality and fast rendering for bitmaps, 2D geometries, and text.

Bitmap allows you to use the Direct2D effects and apply them on images. Following is a list of image effects that can be applied to an image using Bitmap:

- Gaussian Blur
- Sharpen
- Horizontal Smear
- Shadow
- Displacement Map
- Emboss
- Edge Detect
- Sepia

Let us take one of these effects and apply it on an image. The following image shows one of the built-in 2D effects, shadow, presenting the use of Direct2D in Bitmap.



In terms of implementation, Bitmap is first converted to a Direct2D bitmap. Direct2D is then used to manipulate the image by applying the built-in shadow effect using interoperation with Direct3D API. After all the manipulations, the image is loaded back from Direct2D bitmap to C1Bitmap.

To apply shadow effect on an image, you can use the properties of [Shadow](#), [AffineTransform2D](#), and [Composite](#) classes, members of [C1.Util.DX.Direct2D.Effects](#) namespace.

The following steps illustrate applying the 2D shadow effect on an image. This example uses the sample created in the Quick Start.

1. Add relevant namespaces.
    - **Visual Basic**
- ```
Imports C1.Win.Bitmap
```

```
Imports D2D = C1.Util.DX.Direct2D
Imports D3D = C1.Util.DX.Direct3D11
Imports DW = C1.Util.DX.DirectWrite
Imports DXGI = C1.Util.DX.DXGI
```

o C#

```
using C1.Win.Bitmap;
using D2D = C1.Util.DX.Direct2D;
using D3D = C1.Util.DX.Direct3D11;
using DW = C1.Util.DX.DirectWrite;
using DXGI = C1.Util.DX.DXGI;
using C1.Util.DX;
```

2. Create various class objects.

o Visual Basic

```
Private bitmap As C1Bitmap
Private lastGdiBitmap As Bitmap

' device-independent resources
Private d2dFactory As D2D.Factory2
Private dwFactory As DW.Factory

' device resources
Private dxgiDevice As DXGI.Device
Private d2dContext As D2D.DeviceContext1

' Direct2D built-in effects
Private shadow As D2D.Effects.Shadow
Private affineTransform As D2D.Effects.AffineTransform2D
Private composite As D2D.Effects.Composite
```

o C#

```
C1Bitmap bitmap;
Bitmap lastGdiBitmap;

// device-independent resources
D2D.Factory2 d2dFactory;
DW.Factory dwFactory;

// device resources
DXGI.Device dxgiDevice;
D2D.DeviceContext1 d2dContext;

// Direct2D built-in effects
D2D.Effects.Shadow shadow;
D2D.Effects.AffineTransform2D affineTransform;
D2D.Effects.Composite composite;
```

3. Declare constant integers and enumeration.

o Visual Basic

```
Const marginLT As Integer = 20
Const marginRB As Integer = 36
```

```
Public Enum ImageEffect
    Original
    Shadow
End Enum
```

o C#

```
const int marginLT = 20;
const int marginRB = 36;
```

```
public enum ImageEffect
{
    Original,
    Shadow
}
```

4. Load the image in C1Bitmap using stream. For details, see [Quick start](#).

5. Add code to create resources, image source, and associate the image source with the image.

o Visual Basic

```
' create Direct2D and DirectWrite factories
d2dFactory = D2D.Factory2.Create(D2D.FactoryType.SingleThreaded)
dwFactory = DW.Factory.Create(DW.FactoryType.[Shared])
```

```
' create GPU resources
CreateDeviceResources()
```

o C#

```
// create Direct2D and DirectWrite factories
```

```
d2dFactory = D2D.Factory2.Create(D2D.FactoryType.SingleThreaded);
dwFactory = DW.Factory.Create(DW.FactoryType.Shared);
```

```
// create GPU resources
CreateDeviceResources();
```

6. Add code to apply 2D shadow effect.

o **Visual Basic**

```
Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
    UpdateImageSource(ImageEffect.Shadow)
End Sub

Private Sub CreateDeviceResources()
    ' create the Direct3D device
    Dim actualLevel As D3D.FeatureLevel
    Dim d3dContext As D3D.DeviceContext = Nothing
    Dim d3dDevice = New D3D.Device(IntPtr.Zero)
    Dim result = HRESULT.Ok
    For i As Integer = 0 To 1
        ' use WARP if hardware is not available
        Dim dt = If(i = 0, D3D.DriverType.Hardware, D3D.DriverType.Warp)
        result = D3D.D3D11.CreateDevice(Nothing, dt, IntPtr.Zero, _
            D3D.DeviceCreationFlags.BgraSupport Or _
            D3D.DeviceCreationFlags.SingleThreaded, _
            Nothing, 0, D3D.D3D11.SdkVersion, d3dDevice, _
            actualLevel, d3dContext)

        If result.Code <> CInt(&H887A0004UI) Then
            ' DXGI_ERROR_UNSUPPORTED
            Exit For
        End If
    Next
    result.CheckError()
    d3dContext.Dispose()

    ' store the DXGI device (for trimming when the application is being suspended)
    dxgiDevice = d3dDevice.QueryInterface(Of DXGI.Device)()
    d3dDevice.Dispose()

    ' create a RenderTarget (DeviceContext for Direct2D drawing)
    Dim d2dDevice = D2D.Device1.Create(d2dFactory, dxgiDevice)
    Dim rt = D2D.DeviceContext1.Create(d2dDevice, D2D.DeviceContextOptions.None)
    d2dDevice.Dispose()
    rt.SetUnitMode(D2D.UnitMode.Pixels)
    d2dContext = rt

    ' create built-in effects
    shadow = D2D.Effects.Shadow.Create(rt)
    affineTransform = D2D.Effects.AffineTransform2D.Create(rt)
    composite = D2D.Effects.Composite.Create(rt)
End Sub

Private Sub DiscardDeviceResources()
    shadow.Dispose()
    affineTransform.Dispose()
    composite.Dispose()

    dxgiDevice.Dispose()
    d2dContext.Dispose()
End Sub

Private Sub ClearGdiBitmap()
    If lastGdiBitmap IsNot Nothing Then
        PictureBox1.Image = Nothing
        lastGdiBitmap.Dispose()
        lastGdiBitmap = Nothing
    End If
End Sub

Private Sub UpdateImageSource(imageEffect_1 As ImageEffect)
    Dim targetOffset = New Point2F(marginLT, marginLT)
    Dim w As Integer = bitmap.PixelWidth + marginLT + marginRB
    Dim h As Integer = bitmap.PixelHeight + marginLT + marginRB

    ' the render target object
    Dim rt = d2dContext
```



```

' create the target Direct2D bitmap
Dim bpTarget = New D2D.BitmapProperties1 _
    (New D2D.PixelFormat(DXGI.Format.B8G8R8A8_UNorm, D2D.AlphaMode.Premultiplied), _
    CSng(bitmap.DpiX), CSng(bitmap.DpiY), _
    D2D.BitmapOptions.Target Or D2D.BitmapOptions.CannotDraw)
Dim targetBmp = D2D.Bitmap1.Create(rt, New Size2L(w, h), bpTarget)

' associate the target bitmap with render target
rt.SetTarget(targetBmp)

' start drawing
rt.BeginDraw()

' clear the target bitmap
rt.Clear(Nothing)

' convert ClBitmap image to Direct2D image
Dim d2dBitmap = bitmap.ToD2DBitmap1(rt, D2D.BitmapOptions.None)

' apply the effect
Select Case imageEffect_1
    Case ImageEffect.Original
        rt.DrawImage(d2dBitmap, targetOffset)
    Exit Select
    Case ImageEffect.Shadow
        rt.DrawImage(ApplyShadow(d2dBitmap), targetOffset)
    Exit Select
End Select
d2dBitmap.Dispose()

' finish drawing (all drawing commands are executed at that moment)
rt.EndDraw()

' detach and actually dispose the target bitmap
rt.SetTarget(Nothing)

' create a temporary ClBitmap object
Dim outBitmap = New ClBitmap(bitmap.ImagingFactory)

' import the image from Direct2D target bitmap to ClBitmap
outBitmap.Import(targetBmp, rt, New RectL(w, h))
targetBmp.Dispose()

' convert ClBitmap to a System.Drawing.Bitmap
ClearGdiBitmap()
lastGdiBitmap = outBitmap.ToGdiBitmap()
outBitmap.Dispose()

' show the result in the PictureBox
PictureBox1.Image = lastGdiBitmap
End Sub

Private Function ApplyShadow(bitmap As D2D.Bitmap1) As D2D.Effect
    shadow.SetInput(0, bitmap)
    shadow.BlurStandardDeviation = 5.0F
    affineTransform.SetInputEffect(0, shadow)
    affineTransform.TransformMatrix = Matrix3x2.Translation(20.0F, 20.0F)
    composite.SetInputEffect(0, affineTransform)
    composite.SetInput(1, bitmap)
    Return composite
End Function

o C#
private void button2_Click(object sender, EventArgs e)
{
    UpdateImageSource(ImageEffect.Shadow);
}

void CreateDeviceResources()
{
    // create the Direct3D device
    D3D.FeatureLevel actualLevel;
    D3D.DeviceContext d3dContext = null;
    var d3DDevice = new D3D.Device(IntPtr.Zero);

```

```

var result = HRESULT.Ok;
for (int i = 0; i <= 1; i++)
{
    // use WARP if hardware is not available
    var dt = i == 0 ? D3D.DriverType.Hardware : D3D.DriverType.Warp;
    result = D3D.D3D11.CreateDevice
        (null, dt, IntPtr.Zero,
         D3D.DeviceCreationFlags.BgraSupport | D3D.DeviceCreationFlags.SingleThreaded,
         null, 0, D3D.D3D11.SdkVersion, d3dDevice, out actualLevel, out d3dContext);
    if (result.Code != unchecked((int)0x887A0004)) // DXGI_ERROR_UNSUPPORTED
    {
        break;
    }
}
result.CheckError();
d3dContext.Dispose();

// store the DXGI device (for trimming when the application is being suspended)
dxgiDevice = d3dDevice.QueryInterface<DXGI.Device>();
d3dDevice.Dispose();

// create a RenderTarget (DeviceContext for Direct2D drawing)
var d2dDevice = D2D.Device1.Create(d2dFactory, dxgiDevice);
var rt = D2D.DeviceContext1.Create(d2dDevice, D2D.DeviceContextOptions.None);
d2dDevice.Dispose();
rt.SetUnitMode(D2D.UnitMode.Pixels);
d2dContext = rt;

// create built-in effects
shadow = D2D.Effects.Shadow.Create(rt);
affineTransform = D2D.Effects.AffineTransform2D.Create(rt);
composite = D2D.Effects.Composite.Create(rt);
}

void DiscardDeviceResources()
{
    shadow.Dispose();
    affineTransform.Dispose();
    composite.Dispose();

    dxgiDevice.Dispose();
    d2dContext.Dispose();
}

void ClearGdiBitmap()
{
    if (lastGdiBitmap != null)
    {
        pictureBox1.Image = null;
        lastGdiBitmap.Dispose();
        lastGdiBitmap = null;
    }
}

void UpdateImageSource(ImageEffect imageEffect)
{
    var targetOffset = new Point2F(marginLT, marginLT);
    int w = bitmap.PixelWidth + marginLT + marginRB;
    int h = bitmap.PixelHeight + marginLT + marginRB;

    // the render target object
    var rt = d2dContext;

    // create the target Direct2D bitmap
    var bpTarget = new D2D.BitmapProperties1(
        new D2D.PixelFormat(DXGI.Format.B8G8R8A8_UNorm, D2D.AlphaMode.Premultiplied),
        (float)bitmap.DpiX,
        (float)bitmap.DpiY, D2D.BitmapOptions.Target | D2D.BitmapOptions.CannotDraw);
    var targetBmp = D2D.Bitmap1.Create(rt, new Size2L(w, h), bpTarget);

    // associate the target bitmap with render target
    rt.SetTarget(targetBmp);

    // start drawing

```

```
rt.BeginDraw();

// clear the target bitmap
rt.Clear(null);

// convert C1Bitmap image to Direct2D image
var d2dBitmap = bitmap.ToD2DBitmap1(rt, D2D.BitmapOptions.None);

//apply the effect
switch (imageEffect)
{
    case ImageEffect.Original:
        rt.DrawImage(d2dBitmap, targetOffset);
        break;
    case ImageEffect.Shadow:
        rt.DrawImage(ApplyShadow(d2dBitmap), targetOffset);
        break;
}
d2dBitmap.Dispose();

// finish drawing (all drawing commands are executed at that moment)
rt.EndDraw();

// detach and actually dispose the target bitmap
rt.SetTarget(null);

// create a temporary C1Bitmap object
var outBitmap = new C1Bitmap(bitmap.ImagingFactory);

// import the image from Direct2D target bitmap to C1Bitmap
outBitmap.Import(targetBmp, rt, new RectL(w, h));
targetBmp.Dispose();

// convert C1Bitmap to a System.Drawing.Bitmap
ClearGdiBitmap();
lastGdiBitmap = outBitmap.ToGdiBitmap();
outBitmap.Dispose();

// show the result in the PictureBox
pictureBox1.Image = lastGdiBitmap;
}

D2D.Effect ApplyShadow(D2D.Bitmap1 bitmap)
{
    shadow.SetInput(0, bitmap);
    shadow.BlurStandardDeviation = 5f;
    affineTransform.SetInputEffect(0, shadow);
    affineTransform.TransformMatrix = Matrix3x2.Translation(20f, 20f);
    composite.SetInputEffect(0, affineTransform);
    composite.SetInput(1, bitmap);
    return composite;
}
```

Bitmap Samples

With **C1Studio** installer, you get samples that help you understand the product and its implementation better. Bitmap sample is available in the installed folder - **Documents\ComponentOne Samples\WinForms\Bitmap\CS**.

Sample	Description
BitmapSamples	Includes a sample that demonstrates using C1Bitmap to crop an image, distort an image, and apply various transformations such as clipping, flipping, scaling and rotating an image.