ComponentOne

Bitmap for WPF

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)

Table of Contents

Bitmap for WPF

ComponentOne Studio introduces **Bitmap for WPF,** a class library designed to load, save, 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 a wide range of container formats to cater diverse image processing needs.

Help with WPF Edition

For information on installing **ComponentOne Studio WPF Edition**, licensing, technical support, namespaces, and creating a project with the controls, please visit Getting Started with WPF 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 in code.

- **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

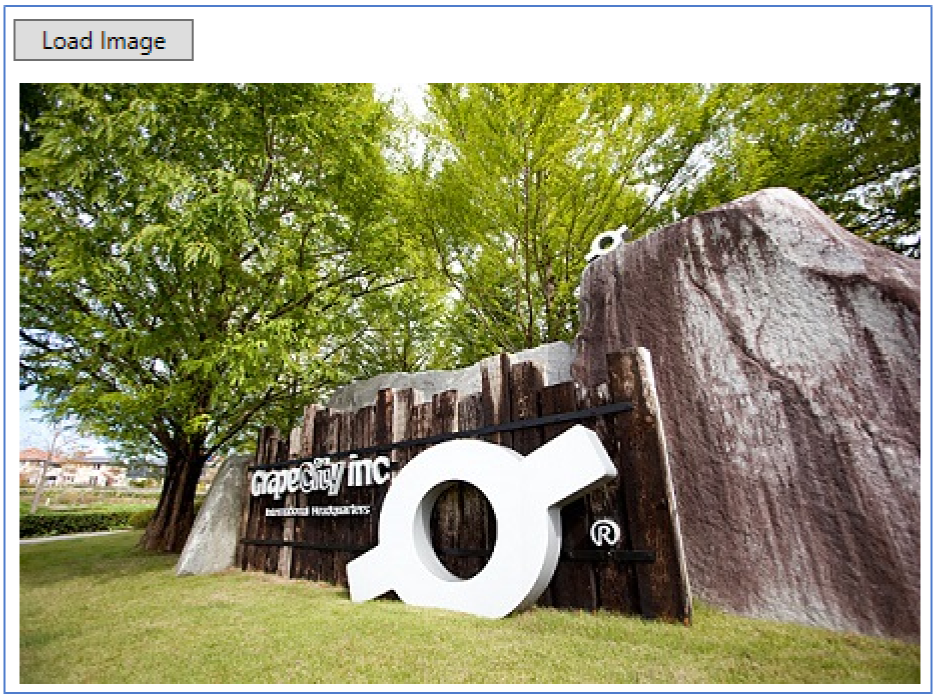
Quick Start

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

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

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 **WPF** application in Visual Studio.
2. Add the following references to your application.
   - C1.WPF.4
   - C1.WPF.Bitmap.4
   - C1.WPF.Automation.4
   - C1.WPF.DX.4
3. In the **Solution Explorer**, right click your project name and select **Add | New Folder** and name it **'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 an **image** control for displaying the sample image onto the MainWindow.
6. Set the **Content** property of the button to a suitable text in XAML view.

Step 2: Adding code to load image using Bitmap

1. Switch to the code view and add the following import statements.
   - **Visual Basic**

```
Imports C1.WPF
Imports C1.WPF.Bitmap
Imports C1.Util.DX
Imports System.Reflection
Imports System.IO
```

   - **C#**

```
using C1.WPF;
using C1.WPF.Bitmap;
using C1.Util.DX;
using System.Reflection;
using System.IO;
```

2. Initialize a bitmap in the MainWindow class.
   - **Visual Basic**

```
'Initialize a bitmap
Private bitmap As C1Bitmap

Public Sub New()

    ' This call is required by the designer.
    InitializeComponent()
    bitmap = New C1Bitmap()
    ' Add any initialization after the InitializeComponent() call.

End Sub
```

   - **C#**

```
//Initialize a bitmap
C1Bitmap bitmap = new C1Bitmap();
```

3. 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 RoutedEventArgs) _
    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 Button_Click(object sender, RoutedEventArgs e)
{
    Assembly asm = typeof(MainWindow).Assembly;
    using (Stream stream =
        asm.GetManifestResourceStream("Bitmap.Resources.GrapeCity.png"))
    {
        bitmap.Load(stream,
            new FormatConverter(PixelFormat.Format32bppPBGRA));
    }

    UpdateImage();
}
```

4. Add the following code to define UpdateImage method for displaying the sample image.
   - **Visual Basic**

```
'Display the loaded image
Private Sub UpdateImage()
    Me.image.Source = bitmap.ToWriteableBitmap()
    Me.image.Width = bitmap.PixelWidth
    Me.image.Height = bitmap.PixelHeight
End Sub
```

- C#

```
//Display the image loaded in bitmap in the image control
private void UpdateImage()
{
    this.image.Source = bitmap.ToWriteableBitmap();
    this.image.Width = bitmap.PixelWidth;
    this.image.Height = bitmap.PixelHeight;
}
```

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 arbitrary 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 MainWindow
    Inherits Window

    'Initializing bitmap as a global variable
    Dim bitmap As New C1Bitmap()

    'Event to load an arbitrary image into the image control on button click
    Private Sub Button_Click(sender As Object, e As RoutedEventArgs)
        Dim ofd = New OpenFileDialog()
        ofd.Filter = "Image Files|*.ico;*.bmp;*.gif;" +
                "*.png;*.jpg;*.jpeg;*.jxr;*.tif;*.tiff"

        If ofd.ShowDialog().Value Then
            bitmap.Load(ofd.FileName,
                        New FormatConverter(PixelFormat.Format32bppPBGRA))
            Image.Source = bitmap.ToWriteableBitmap()
            Image.Width = bitmap.PixelWidth
            Image.Height = bitmap.PixelHeight
        End If
    End Sub

    'Event to save the image to a file on button click
    Private Sub Button_Click_1(sender As Object, e As RoutedEventArgs)
        Dim sfd As New SaveFileDialog()
        sfd.Filter = "Png Files (*.png)|*.png"
        sfd.CheckPathExists = True

        If sfd.ShowDialog().Value Then
            bitmap.Save(sfd.FileName, ContainerFormat.Png)
        End If
    End Sub
End Class
```

- **C#**

```
public partial class MainWindow : Window
{
    //Defining a global variable for bitmap
    C1Bitmap bitmap;

    public MainWindow()
    {
        InitializeComponent();

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

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

        if (ofd.ShowDialog().Value)
        {
            bitmap.Load(ofd.FileName, new
                FormatConverter(PixelFormat.Format32bppPBGRA));
            image.Source = bitmap.ToWriteableBitmap();
            image.Width = bitmap.PixelWidth;
            image.Height = bitmap.PixelHeight;
        }
    }

    //Event to save the image to a file on button click
    private void Button_Click_1(object sender, RoutedEventArgs e)
    {
        SaveFileDialog sfd = new SaveFileDialog();
        sfd.Filter = "Png Files (*.png)|*.png";
        sfd.CheckPathExists = true;

         if (sfd.ShowDialog().Value)
         {
             bitmap.Save(sfd.FileName, ContainerFormat.Png);
         }

    }
}
```

Applying Transformations

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

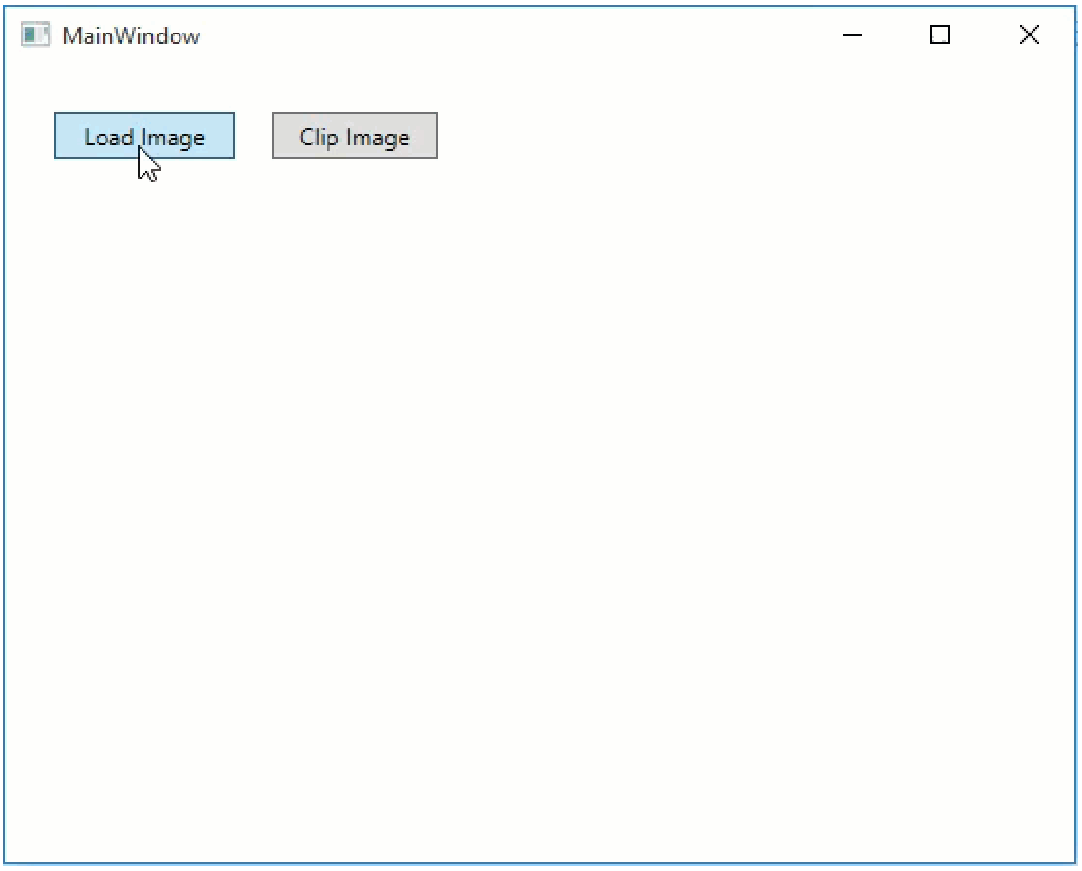
Clipping an image
    Learn how to implement clipping in code.
Flipping an image
    Learn how to implement flipping in code.
Rotating an image
    Learn how to implement rotating in code.
Scaling an image

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 through 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.IO
```

   - **C#**

```
using System.IO;
```

2. Initialize a rectangle and a point as global variables in Form1 class.
   - **Visual Basic**

```
'Initialize variables
Dim start As Point
Dim selection As Rect
Dim dragHelper As C1DragHelper
```

   - **C#**

```
//Initialize variables
Point start;
Rect selection;
C1DragHelper dragHelper;
```

3. Initialize a variable to control drag gestures and subscribe a drag event in the MainWindow constructor.
   - **Visual Basic**

```
'Initialize drag helper for drag gestures
```

```
dragHelper = New C1DragHelper(image)

'Subscribe a drag event
dragHelper.DragDelta += dragHelper_DragDelta()
```

- **C#**

```
//Initialize drag helper for drag gestures
dragHelper = new C1DragHelper(image);

//Subscribe a drag event
dragHelper.DragDelta += dragHelper_DragDelta;
```

4. Add the following code to apply clipper transformation.
   - **Visual Basic**

```
'Transform method to apply transformation
Private Sub ApplyTransform(t As BaseTransform)
    Dim newBitmap = bitmap.Transform(t)
    bitmap.Dispose()
    bitmap = newBitmap
    UpdateImage()
End Sub

'Event to apply clipper transformation on button click
Private Sub Button_Click(sender As Object, e As RoutedEventArgs)
    Dim cropRect = DirectCast(selection, RectD).Round()
    ApplyTransform(New Clipper(New ImageRect(cropRect)))
End Sub
```

   - **C#**

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

//Event to apply clipper transformation on button click
private void Button_Click_1(object sender, RoutedEventArgs e)
{
    var cropRect = ((RectD)selection).Round();
    ApplyTransform(new Clipper(new ImageRect(cropRect)));
}
```

5. Add the following code to select a portion of the image to be clipped.
   - **Visual Basic**

```
'Events to selection a portion of image
Private Sub dragHelper_DragDelta(sender As Object, e As C1DragDeltaEventArgs)
    Dim pos = e.GetPosition(image)
    pos = New Point(Math.Max(0, Math.Min(pos.X, bitmap.PixelWidth)),
                    Math.Max(0, Math.Min(pos.Y, bitmap.PixelHeight)))

    selection = New Rect(Math.Round(Math.Min(start.X, pos.X)),
                         Math.Round(Math.Min(start.Y, pos.Y)),
                         Math.Round(Math.Abs(start.X - pos.X)),
                         Math.Round(Math.Abs(start.Y - pos.Y)))
End Sub

Private Sub image_MouseLeftButtonDown(sender As Object, e As MouseButtonEventArgs) _
    Handles image.MouseLeftButtonDown
    MyBase.OnMouseLeftButtonDown(e)
    Dim pos = e.GetPosition(image)
    start = New Point(Math.Max(0, Math.Min(pos.X, bitmap.PixelWidth)),
                      Math.Max(0, Math.Min(pos.Y, bitmap.PixelHeight)))
End Sub
```

```
Private Sub image_MouseLeftButtonUp(sender As Object, e As MouseButtonEventArgs) _
    Handles image.MouseLeftButtonUp
    MyBase.OnMouseLeftButtonUp(e)
    Dim pt = e.GetPosition(image)
    If Math.Abs(pt.X - start.X) < 4 AndAlso Math.Abs(pt.Y - start.Y) < 4 Then
        selection = New Rect(0, 0, bitmap.PixelWidth, bitmap.PixelHeight)
    End If
End Sub
```

- **C#**

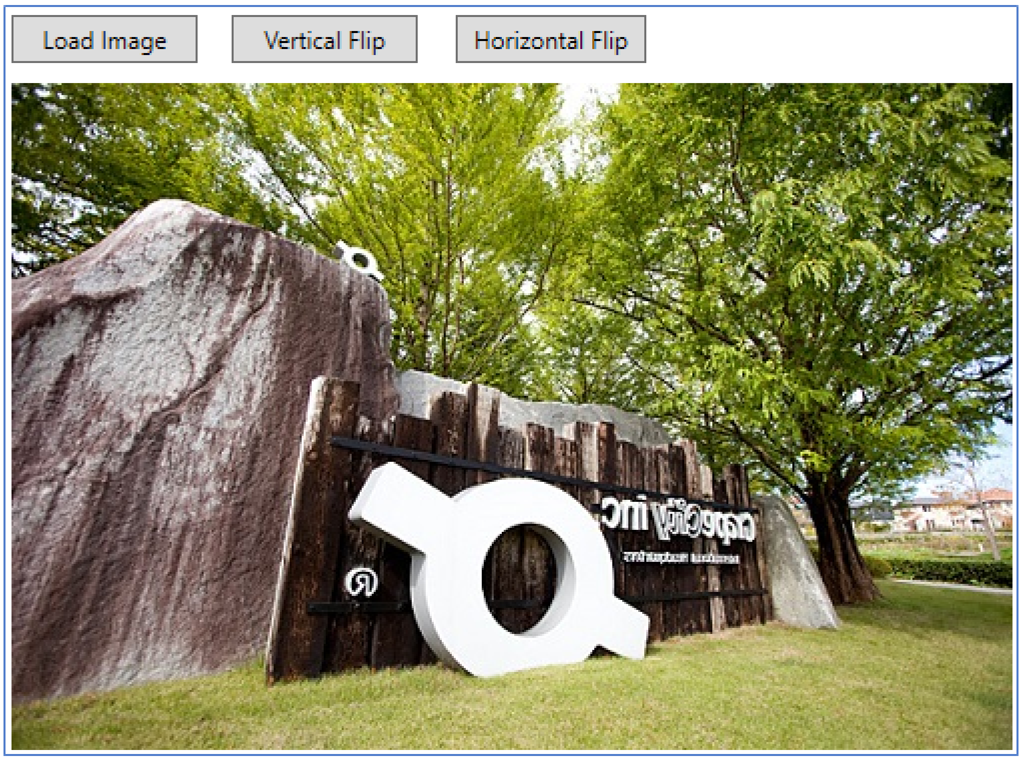
```
//Events to select a portion of image from the image
private void image_MouseLeftButtonDown(object sender,
    System.Windows.Input.MouseButtonEventArgs e)
{
   base.OnMouseLeftButtonDown(e);
    var pos = e.GetPosition(image);
    start = new Point(
    Math.Max(0, Math.Min(pos.X, bitmap.PixelWidth)),
    Math.Max(0, Math.Min(pos.Y, bitmap.PixelHeight)));
}

private void image_MouseLeftButtonUp(object sender,
    System.Windows.Input.MouseButtonEventArgs e)
{
    base.OnMouseLeftButtonUp(e);
    var pt = e.GetPosition(image);
    if (Math.Abs(pt.X - start.X) < 4 && Math.Abs(pt.Y - start.Y) < 4)
    {
        selection = new Rect(0, 0, bitmap.PixelWidth, bitmap.PixelHeight);
    }
}

void dragHelper_DragDelta(object sender, C1DragDeltaEventArgs e)
{
    var pos = e.GetPosition(image);
    pos = new Point(Math.Max(0, Math.Min(pos.X, bitmap.PixelWidth)),
        Math.Max(0, Math.Min(pos.Y, bitmap.PixelHeight)));

    selection = new Rect(
        Math.Round(Math.Min(start.X, pos.X)),
        Math.Round(Math.Min(start.Y, pos.Y)),
        Math.Round(Math.Abs(start.X - pos.X)),
        Math.Round(Math.Abs(start.Y - pos.Y)));
}
```

6. Press F5 to run the application and click 'Load Image' button to load an image.
7. 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 as well as horizontally. To produce a flipped image using Bitmap, you can set the TransformOptions property of the FlipRotator class. The TransformOptions property can be set through TransformOptions enumeration in code.

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
    UpdateImage()
End Sub

'Event to flip the image vertically on button click
Private Sub Button_Click_1(sender As Object, e As RoutedEventArgs)
    ApplyTransform(New FlipRotator(TransformOptions.FlipVertical))
End Sub

'Event to flip the image horizontally on button click
Private Sub Button_Click_2(sender As Object, e As RoutedEventArgs)
    ApplyTransform(New FlipRotator(TransformOptions.FlipHorizontal))
End Sub
```

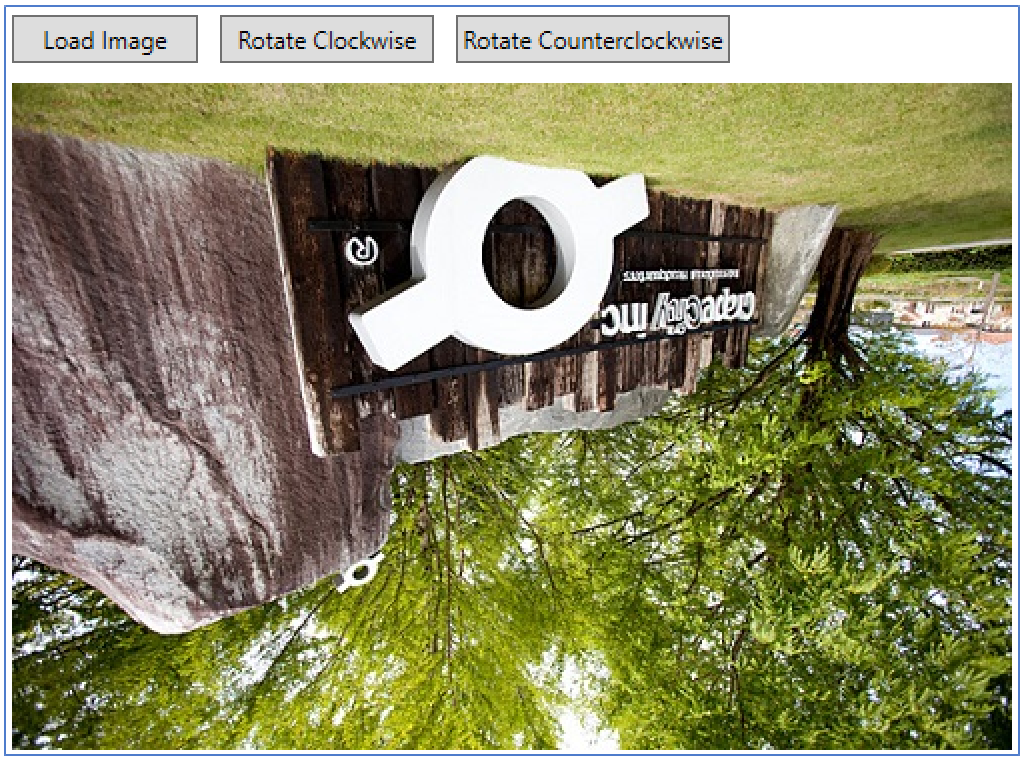
- **C#**

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

//Event to flip the image vertically on button click
private void Button_Click_1(object sender, RoutedEventArgs e)
{
    ApplyTransform(new FlipRotator(TransformOptions.FlipVertical));
}

//Event to flip the image horizontally on button click
private void Button_Click_2(object sender, RoutedEventArgs 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 TransformOptions property can be set through the TransformOptions enumeration.

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



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
    UpdateImage()
End Sub

'Event to rotate the image in clockwise direction on button click
Private Sub Button_Click_1(sender As Object, e As RoutedEventArgs)
    ApplyTransform(New FlipRotator(TransformOptions.Rotate180))
End Sub

'Event to rotate the image in counterclockwise direction on button click
Private Sub Button_Click_2(sender As Object, e As RoutedEventArgs)
```

```
    ApplyTransform(New FlipRotator(TransformOptions.Rotate270))
End Sub
```

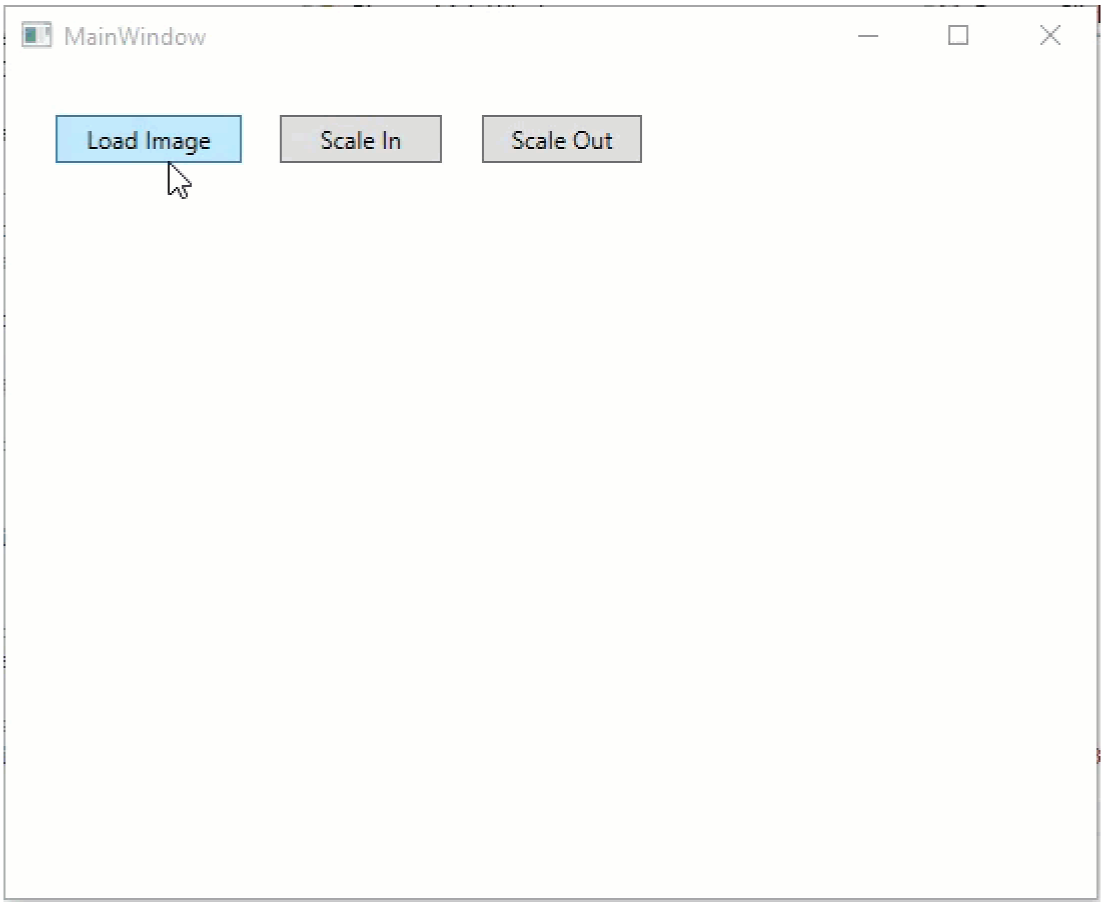
- **C#**

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

//Event to rotate the image in clockwise direction on button click
private void Button_Click_1(object sender, RoutedEventArgs e)
{
    ApplyTransform(new FlipRotator(TransformOptions.Rotate180));
}

//Event to rotate the image in counterclockwise direction on button click
private void Button_Click_2(object sender, RoutedEventArgs 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.

- **Visual Basic**

```
Private Sub ApplyTransform(t As BaseTransform)
    Dim newBitmap = bitmap.Transform(t)
    bitmap.Dispose()
    bitmap = newBitmap
    UpdateImage()
End Sub

'Event to scale in the image on button click
Private Sub Button_Click_1(sender As Object, e As RoutedEventArgs)
    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, InterpolationMode.HighQualityCubic))
    End If
End Sub

'Event to scale out the image on button click
Private Sub Button_Click_2(sender As Object, e As RoutedEventArgs)
    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, InterpolationMode.HighQualityCubic))
End Sub
```

- **C#**

```
void ApplyTransform(BaseTransform t)
```

```
{
    var newBitmap = bitmap.Transform(t);
    bitmap.Dispose();
    bitmap = newBitmap;
    UpdateImage();
}

//Event to scale in the image on button click
private void Button_Click_1(object sender, RoutedEventArgs 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, InterpolationMode.HighQualityCubic));
    }
}

//Event to scale out the image on button click
private void Button_Click_2(object sender, RoutedEventArgs e)
{
    int px = (int)(bitmap.PixelWidth * 1.6f + 0.5f);
    int py = (int)(bitmap.PixelHeight * 1.6f + 0.5f);
    ApplyTransform(new Scaler(px, py, 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.

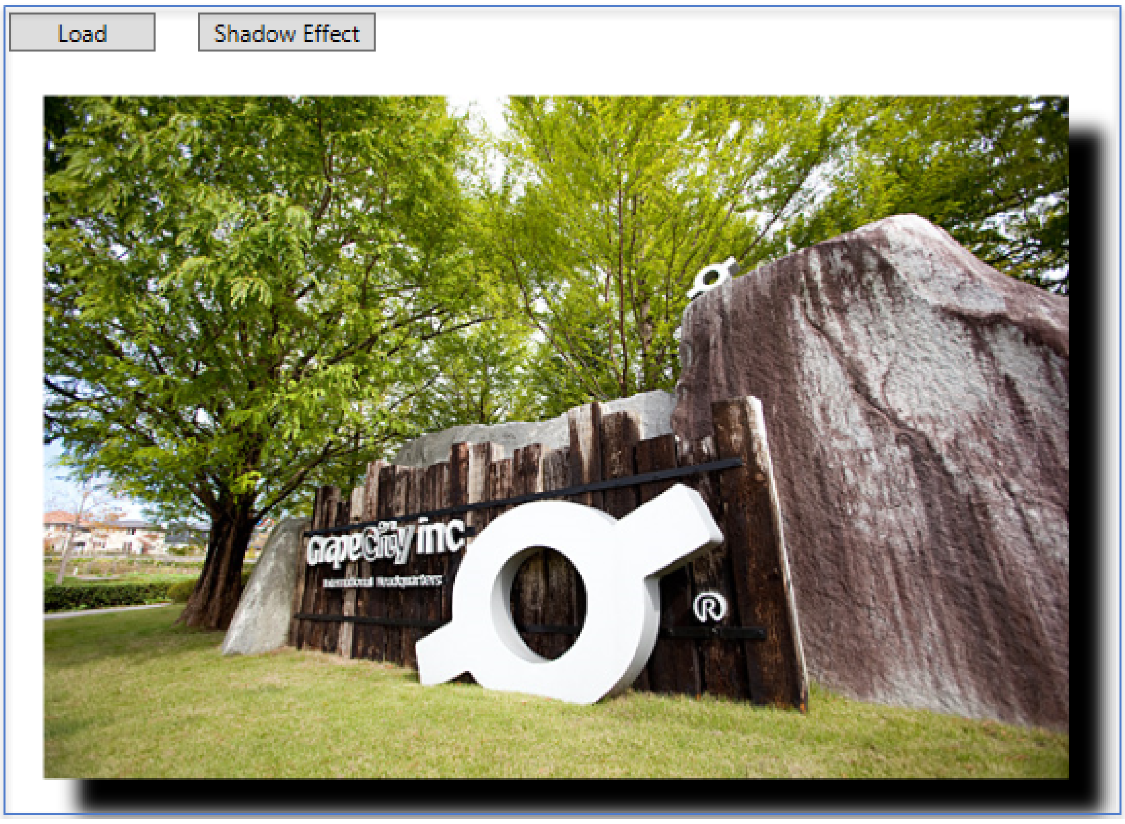
Applying Direct2D Effects

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

   - **C#**

```
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.
   - **Visual Basic**

```
Private bitmap As C1Bitmap

' 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
```

   - **C#**

```
C1Bitmap bitmap;

// 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.
   - **Visual Basic**

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

Public Enum ImageEffect
    Original
    Shadow
End Enum
```

   - **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.
    - **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()
```

   - **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.
    - **Visual Basic**

```
Private Sub Button_Click(sender As Object, e As RoutedEventArgs)

    UpdateImageSource(ImageEffect.Shadow)
End Sub

Private Sub CreateDeviceResources()
    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 UpdateImageSource(imageEffect__1 As ImageEffect)
        ' some effects can change pixels outside the bounds of the source
        ' image, so we need a margin to make those pixels visible
        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 C1Bitmap image to Direct2D image
        Dim d2dBitmap = bitmap.ToD2DBitmap1(rt, D2D.BitmapOptions.None)

        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()

        If Not rt.EndDraw(True) Then
            targetBmp.Dispose()

            ' try recreate device resources if old GPU device was removed
            DiscardDeviceResources()
            CreateDeviceResources()
            Return
        End If

        ' detach the target bitmap
        rt.SetTarget(Nothing)

        ' create a temporary C1Bitmap object
        Dim 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 WriteableBitmap, then use it as an image source
        image.Source = outBitmap.ToWriteableBitmap()
        outBitmap.Dispose()

    End Sub

    Private Sub DiscardDeviceResources()
        shadow.Dispose()
        affineTransform.Dispose()
        composite.Dispose()
        dxgiDevice.Dispose()
        d2dContext.Dispose()
    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
```

- **C#**

```
    private void Button_Click(object sender, RoutedEventArgs e)
    {
        UpdateImageSource(ImageEffect.Shadow);
    }

    private void CreateDeviceResources()
    {
        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 UpdateImageSource(ImageEffect imageEffect)
    {
        //some effects can change pixels outside the bounds of the source
        //image, so we need a margin to make those pixels visible
        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);

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

        if (!rt.EndDraw(true))
        {
            targetBmp.Dispose();

            // try to recreate the device resources if old GPU device was removed
            DiscardDeviceResources();
            CreateDeviceResources();
            return;
        }

        //detach 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 WriteableBitmap, then use it as an image source
        image.Source = outBitmap.ToWriteableBitmap();
        outBitmap.Dispose();

    }

    private void DiscardDeviceResources()
    {
        shadow.Dispose();
        affineTransform.Dispose();
        composite.Dispose();
        dxgiDevice.Dispose();
        d2dContext.Dispose();
    }

    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\WPF\C1.WPF.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.