ComponentOne

Bitmap for UWP

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

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 $2 5 (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

Bitmap for UWP

ComponentOne Studio introduces **Bitmap for UWP,** a class library designed to load, save, and transform images. Using **Bitmap**, you can scale, clip, flip, or rotate the image, change the pixel format, or apply any arbitrary combination of these transformations on an image. **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 UWP Edition

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

Key Features

Bitmap provides number of supported features for handling images. Besides simple image display and save, you can use Bitmap for various purposes listed below:

- **Load Images**
  Bitmap loads or imports image with a variety of container formats including BMP, PNG, JPEG, JPEG-XR, TIFF, ICO, and GIFF (Single frame). The image can be loaded from a file, stream, from another bitmap object, or imported from a byte array. Bitmap also allows loading several images, one by one, into the same instance of **C1Bitmap**.

- **Save Images**
  As with loading, the image from C1Bitmap can be saved to a StorageFile or **IOutputStream** and to **System.IO.Stream**. C1Bitmap provides a specific **SaveAs** method for each of the supported container formats, such as BMP, PNG, JPEG, JPEG-XR, TIFF, and GIFF (Single frame) except ICO as saving an image in this format is not supported.

- **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 using **Transform** method.

- **Apply Direct2D Effects**
  Bitmap allows you to apply Direct2D effects on an image for superior imaging effects.

- **Convert to/from Platform-Specific Bitmap Objects**
  One of the key ability of **Bitmap** is its support for conversion to/from the platform-specific bitmap objects. For example, in a Windows Store app you can easily convert **C1Bitmap** to a **WriteableBitmap** or **SoftwareBitmap**. Also, you can import, for example, a **WriteableBitmap** as fragment into the existing **C1Bitmap** or create a new **C1Bitmap** from the given instance of **SoftwareBitmap**.

Object Model Summary

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

C1Bitmap
Properties: ContainerFormat, HasImage, HasMetadata, ImagingFactory, IsDisposed, NativeBitmap, PixelFormat, PixelHeight, PixelWidth **Methods:** Dispose, Import, Load, LoadAsync, LoadMetadata, Save, SaveAsBmp, SaveAsGif, SaveAsJpeg, SaveAsPng, SaveAsTiff, SaveAsync, ToSoftwareBitmap, ToWritableBitmap, Transform
Clipper
Property: ImageRect
FlipRotator
Property: TransformOptions
FormatConverter
Properties: DestinationFormat, Palette, PaletteTranslate
Scaler
Properties: DestinationHeight, DestinationWidth, InterpolationMode

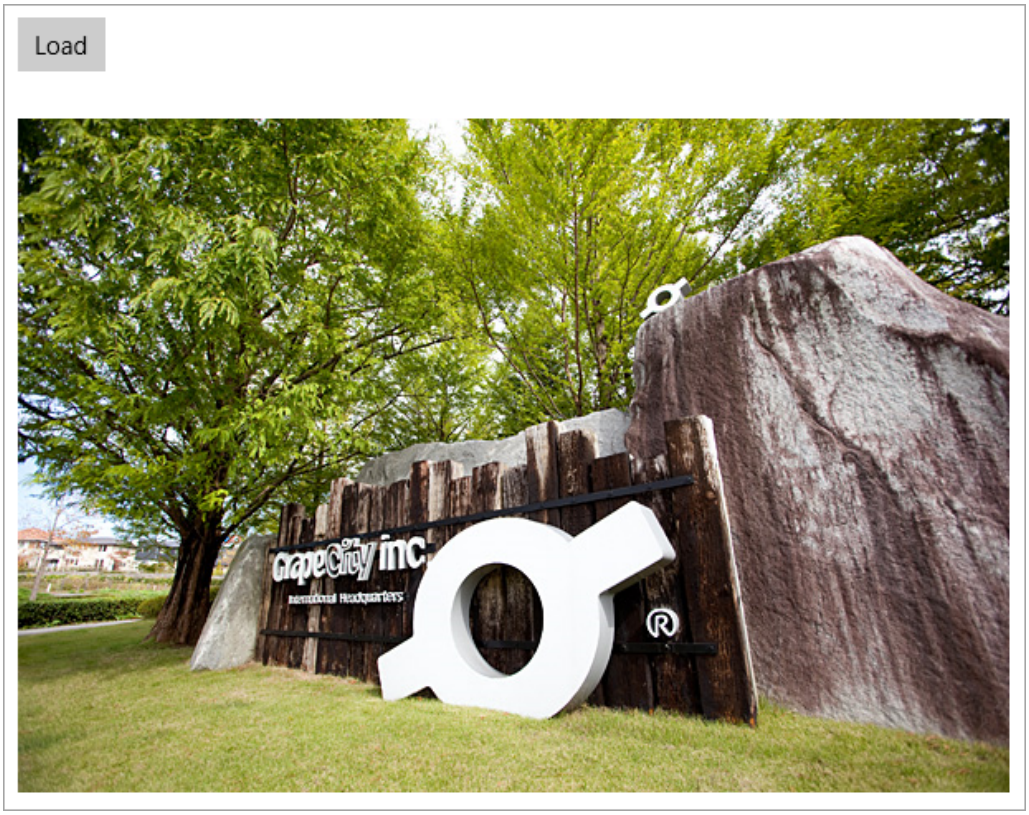
Quick Start

The quick start guide familiarizes you with loading an image in **Bitmap**. You begin with creating a UWP application in Visual Studio, adding C1.UWP.Bitmap reference (dll), Image control, and a button to load an image from stream in **C1Bitmap** object.

To create a simple UWP application for loading an image in C1Bitmap object, follow these steps:

1. **Setting up the Application**
2. **Loading an Image into C1Bitmap**

The following image shows how the application displays an image loaded from a stream into **C1Bitmap**:



Setting up the Application

To set up the application, follow these steps:

1. Create a new project and select **Blank App** (Universal Windows) in **Visual Studio**.
2. Add the following references to the application.
   - C1.UWP.Bitmap
   - C1.UWP.DX
3. In the **Solution Explorer**, right click your project name and select **Add | New Folder**. Name the newly added folder. In our case, we have named the new folder as **Resources**.
4. Add a sample image to the Resources folder and set its **Build Action** property to **Embedded Resource** from the **Properties** window.
5. Add a standard **Button** control for loading a sample image and **Image** control, named img, for displaying the sample image.
6. Set the **Content** property of the button as **Load Image**.

Back to Top

Loading an Image into C1Bitmap

To load the image in **C1Bitmap**, follow these steps:

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

   Visual Basic

   ```
   Imports C1.Xaml.Bitmap
   Imports System.Reflection
   Imports Windows.Graphics.Imaging
   ```

   C#

   ```
   using C1.Xaml.Bitmap;
   using System.Reflection;
   using System.Threading.Tasks;
   using Windows.Graphics.Imaging;
   using Windows.UI.Xaml.Media.Imaging;
   ```

2. Create the following class objects.

   Visual Basic

   ```
   Dim btmp As C1Bitmap
   Dim sb As SoftwareBitmap
   Dim sbs As SoftwareBitmapSource
   ```

   C#

   ```
   C1Bitmap btmp;
   SoftwareBitmap sb;
   SoftwareBitmapSource sbs;
   ```

3. Add the following code in the class constructor to initialize bitmap and SoftwareBitmapSource.

   Visual Basic

   ```
   btmp = New C1Bitmap()
   sbs = New SoftwareBitmapSource()
   ```

   C#

   ```
   btmp = new C1Bitmap();
   sbs = new SoftwareBitmapSource();
   ```

4. Add the following code to the **btnLoad_Click** event to load the image in **C1Bitmap** using stream:

   Visual Basic

   ```
   Dim asm As Assembly = GetType(MainPage).GetTypeInfo().Assembly
   Using stream As Stream =
   asm.GetManifestResourceStream("BitmapUWP_VB.GrapeCity.jpg")
       btmp.Load(stream, New FormatConverter(PixelFormat.Format32bppPBGRA))
   End Using
   Await UpdateImageSource()
   ```

   C#

   ```
   Assembly asm = typeof(MainPage).GetTypeInfo().Assembly;
   using (Stream stream =
   asm.GetManifestResourceStream("BitmapUWP.Resources.GrapeCity.jpg"))
   {
   ```

```
    btmp.Load(stream, new FormatConverter(PixelFormat.Format32bppPBGRA));
}
await UpdateImageSource();
```

> **Note:** To call an asynchronous method on an event, we have used async keyword and await operator.

5. Create a method, named **UpdateImageSource**, to create a **SoftwareBitmap**, set the source SoftwareBitmap and assign it to the image source:

Visual Basic

```
Private Async Function UpdateImageSource() As Task
    sb = btmp.ToSoftwareBitmap()
    Await sbs.SetBitmapAsync(sb)
    img.Source = sbs
End Function
```

C#

```
private async Task UpdateImageSource()
{
    sb = btmp.ToSoftwareBitmap();
    await sbs.SetBitmapAsync(sb);
    img.Source = sbs;
}
```

Back to Top

Features

Bitmap Features comprises all the features available in Bitmap control.

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 allows loading an image in C1Bitmap object using the Load method of the **C1Bitmap** class. With **Load** and LoadAsync method overloads, the image can be loaded from StorageFile or IInputStream, or from System.IO.Stream. **Bitmap** also allows loading image metadata, which can be used to determine size of the image, its resolution (DPI), or pixel format. Additionally, several images can be loaded or imported, one by one, into the same instance of **C1Bitmap**.

Like loading, the image from **C1Bitmap** can be saved to a StorageFile or IOutputStream, and to System.IO.Stream. The Bitmap provides general Save methods in **C1Bitmap** class, that accept the container format as an argument. It also provides a specific **SaveAs** method for each of the supported container formats which has the corresponding encoder.

Here, we discuss loading an arbitrary image from a file. To load an image from a stream, refer to Quick Start.

The following steps illustrate loading an arbitrary image from a file. This code uses **LoadAsync** method to load an image from a StorageFile.

1. Create and initialize the following class objects.

   Visual Basic

   ```
   Dim btmp As New C1Bitmap()
   Dim sb As SoftwareBitmap
   Dim sbs As SoftwareBitmapSource
   ```

   C#

   ```
   C1Bitmap btmp = new C1Bitmap();
   SoftwareBitmap sb;
   SoftwareBitmapSource sbs;
   ```

2. Add the following code to load an image from a StorageFile.

   Visual Basic

   ```
   Dim picker = New FileOpenPicker()

   picker.FileTypeFilter.Add(".ico")
   picker.FileTypeFilter.Add(".bmp")
   picker.FileTypeFilter.Add(".gif")
   picker.FileTypeFilter.Add(".png")
   picker.FileTypeFilter.Add(".jpg")
   picker.FileTypeFilter.Add(".jpeg")
   picker.FileTypeFilter.Add(".jxr")
   picker.FileTypeFilter.Add(".tif")
   picker.FileTypeFilter.Add(".tiff")

   Dim file As StorageFile = Await picker.PickSingleFileAsync()
   ```

```
If file IsNot Nothing Then
    Await btmp.LoadAsync(file, New
FormatConverter(PixelFormat.Format32bppPBGRA))
    Await UpdateImageSource()
End If
```

C#

```
var picker = new FileOpenPicker();

picker.FileTypeFilter.Add(".ico");
picker.FileTypeFilter.Add(".bmp");
picker.FileTypeFilter.Add(".gif");
picker.FileTypeFilter.Add(".png");
picker.FileTypeFilter.Add(".jpg");
picker.FileTypeFilter.Add(".jpeg");
picker.FileTypeFilter.Add(".jxr");
picker.FileTypeFilter.Add(".tif");
picker.FileTypeFilter.Add(".tiff");

StorageFile file = await picker.PickSingleFileAsync();

if (file != null)
{
        await btmp.LoadAsync(file, new
FormatConverter(PixelFormat.Format32bppPBGRA));
        await UpdateImageSource();
}
```

Note: To call an asynchronous method on an event, we have used async keyword and await operator.

3. Create a method, named **UpdateImageSource**, to create a SoftwareBitmap, set the source SoftwareBitmap and assign it to the image source.

Visual Basic

```
Private Async Function UpdateImageSource() As Task
    sb = btmp.ToSoftwareBitmap()
    sbs = New SoftwareBitmapSource()
    Await sbs.SetBitmapAsync(sb)
    img.Source = sbs
End Function
```

C#

```
async Task UpdateImageSource()
{
    sb = btmp.ToSoftwareBitmap();
    sbs = new SoftwareBitmapSource();
    await sbs.SetBitmapAsync(sb);
    img.Source = sbs;
}
```

4. Save the loaded arbitrary image using the following code. The SaveAsPngAsync method is used here to save the image to a StorageFile in PNG format.

Visual Basic

```
Dim picker = New FileSavePicker()
picker.FileTypeChoices.Add("png", New List(Of String)() From {
    ".png"
})
```

```
picker.DefaultFileExtension = ".png"

Dim file As StorageFile = Await picker.PickSaveFileAsync()

If file IsNot Nothing Then
    Await btmp.SaveAsPngAsync(file, Nothing)
    Dim md As New MessageDialog("Your file have been saved.")

    Await md.ShowAsync()
End If

btmp.Dispose()
```

C#

```
var picker = new FileSavePicker();
picker.FileTypeChoices.Add("png", new List<string> { ".png" });
picker.DefaultFileExtension = ".png";

StorageFile file = await picker.PickSaveFileAsync();

if (file != null)
{
    await btmp.SaveAsPngAsync(file, null);
    MessageDialog md = new MessageDialog("Your file have been saved.");
    await md.ShowAsync();
}
btmp.Dispose();
```

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

Clipping is cropping a portion of an image. Bitmap allows clipping an image with the Clipper class. To clip the source image and load a small fragment instead of the whole image, You can pass the Clipper transformation using the **Clipper** class.

The GIF given below shows clipping an image.

The following code implements clipping of an image on a button's click event. The example uses the sample created in Quick Start.

Visual Basic

```
Private Async Function UpdateImageSource() As Task
    sb = btmp.ToSoftwareBitmap()
    sbs = New SoftwareBitmapSource()
    Await sbs.SetBitmapAsync(sb)
    img.Source = sbs

    img.Width = btmp.PixelWidth
    img.Height = btmp.PixelHeight
End Function

Private Async Function ApplyTransform(t As BaseTransform) As Task
    Dim bm = btmp.Transform(t)
    btmp.Dispose()
    btmp = bm

    Await UpdateImageSource()
End Function

Private Async Sub btnCrop_Click(sender As Object, e As RoutedEventArgs)
    Dim cropRect As New ImageRect(150, 100, 300, 250)
    Await ApplyTransform(New Clipper() With {.ImageRect = cropRect})
End Sub
```

C#

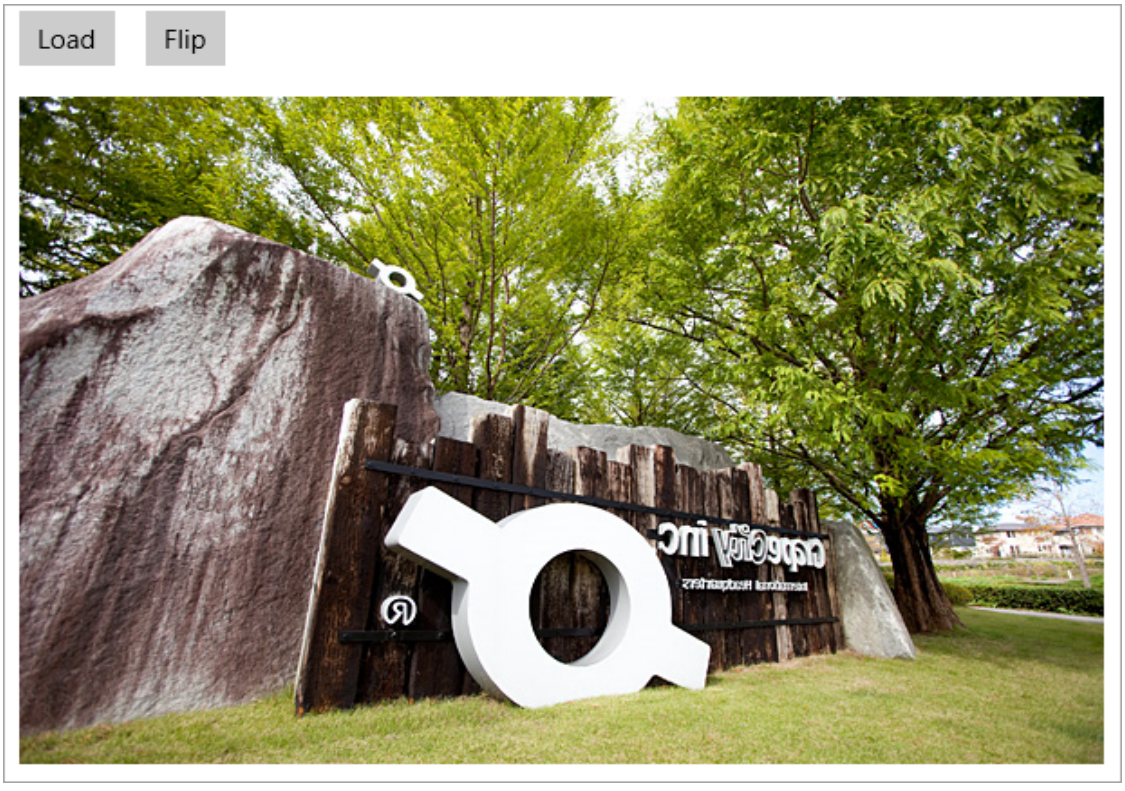
```
private async Task UpdateImageSource()
{
    sb = btmp.ToSoftwareBitmap();
```

```
    sbs = new SoftwareBitmapSource();
    await sbs.SetBitmapAsync(sb);
    img.Source = sbs;

    img.Width = btmp.PixelWidth;
    img.Height = btmp.PixelHeight;
}

private async Task ApplyTransform(BaseTransform t)
{
    var bm = btmp.Transform(t);
    btmp.Dispose();
    btmp = bm;

    await UpdateImageSource();
}

private async void btnClip_Click(object sender, RoutedEventArgs e)
{
    Rect select;
    var cropRect = ((RectD)select).Round();
    await ApplyTransform(new Clipper { ImageRect = new ImageRect(150, 100, 300, 250)
});
}
```

Flipping an Image

Flipping is creating a mirror image of the original image, vertically or horizontally. Bitmap allows both, flipping an image horizontally as well as vertically, using the TransformOptions property of the FlipRotator class. The **TransformOptions** property accepts value from the TransformOptions enum to set the transformation options.

The image given below shows a horizontally flipped image.



The following code implements flipping on an image horizontally on a button's click event. This example uses the sample created in the Quick Start.

Visual Basic

```
Private Async Function UpdateImageSource() As Task
    Dim sb As SoftwareBitmap = btmp.ToSoftwareBitmap()
    Await sbs.SetBitmapAsync(sb)
    img.Source = sbs
End Function

Private Async Function ApplyTransform(t As BaseTransform) As Task
    Dim bm = btmp.Transform(t)
    btmp.Dispose()
    btmp = bm
    Await UpdateImageSource()
End Function
Private Async Sub btnFlip_Click(sender As Object, e As RoutedEventArgs)
    Await ApplyTransform(New FlipRotator(TransformOptions.FlipHorizontal))
End Sub
```

C#

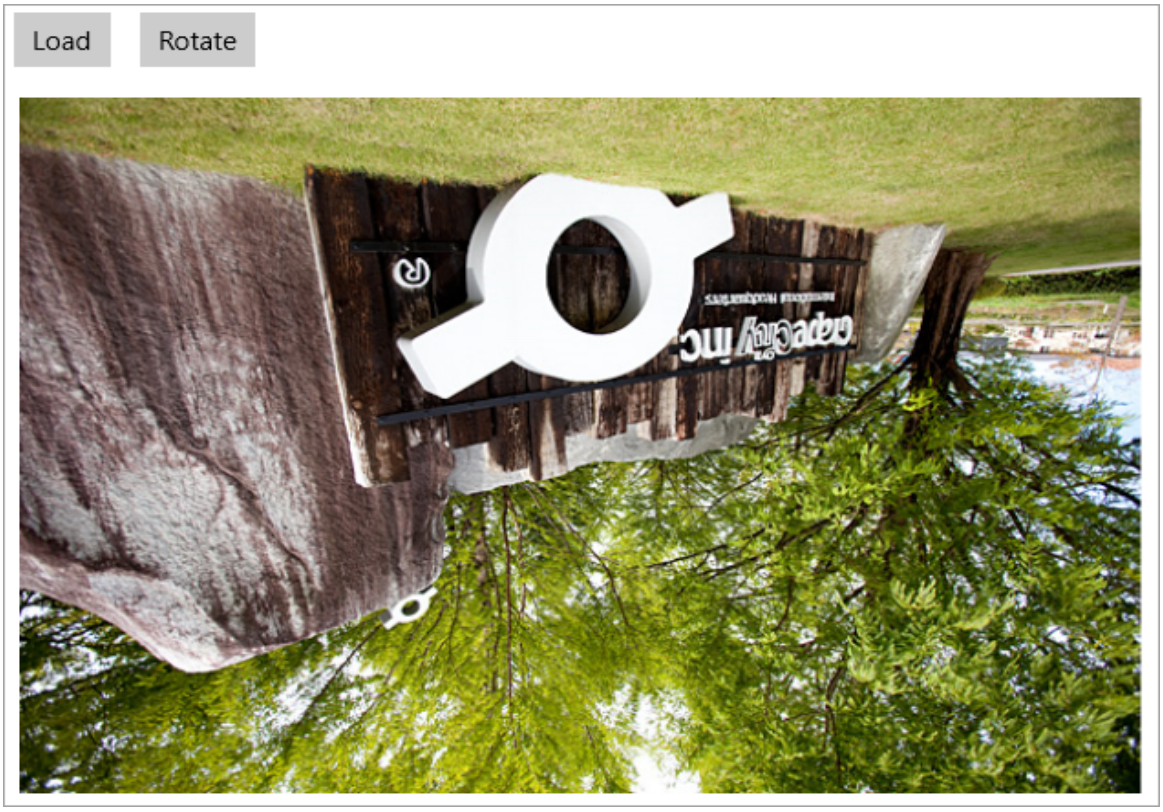
```
private async Task ApplyTransform(BaseTransform t)
{
    var bm = btmp.Transform(t);
    btmp.Dispose();
    btmp = bm;
    await UpdateImageSource();
}

private async void btnFlip_Click(object sender, RoutedEventArgs e)
{
    await ApplyTransform(new FlipRotator(TransformOptions.FlipHorizontal));
}
```

Similarly, you can flip an image vertically using **FlipVertical** value of the **TransformOptions** enum.

Rotating an Image

Bitmap provides the flexibility to rotate images to different angles, 90 degrees, 180 degrees, and 270 degrees in clockwise direction. TransformOptions property of FlipRotator class provided by C1Bitmap can be used to rotate an image. The **TransformOptions** property accepts value from the TransformOptions enum to set the transformation options.

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



The following code implements rotating on an image by 180 degrees in clockwise direction on a button's click event. This example uses the sample created in the Quick Start.

Visual Basic

```
Private Async Function UpdateImageSource() As Task
    Dim sb As SoftwareBitmap = btmp.ToSoftwareBitmap()
    Await sbs.SetBitmapAsync(sb)
    img.Source = sbs
End Function

Private Async Function ApplyTransform(t As BaseTransform) As Task
    Dim bm = btmp.Transform(t)
    btmp.Dispose()
    btmp = bm
    Await UpdateImageSource()
End Function

Private Async Sub btnRotate_Click(sender As Object, e As RoutedEventArgs)
    Await ApplyTransform(New FlipRotator(TransformOptions.Rotate180))
End Sub
```

C#

```
private async Task ApplyTransform(BaseTransform t)
{
    var bm = btmp.Transform(t);
    btmp.Dispose();
    btmp = bm;
    await UpdateImageSource();
}

private async void btnRotate_Click(object sender, RoutedEventArgs e)
```

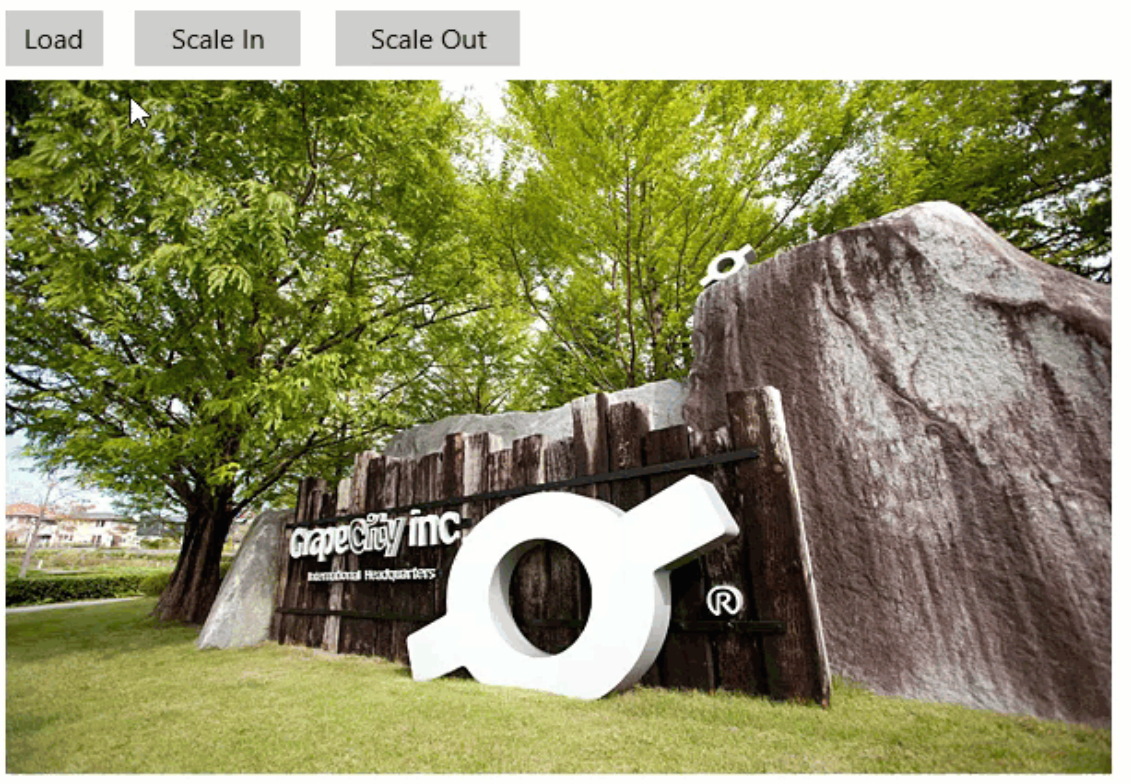
```
{
    await ApplyTransform(new FlipRotator(TransformOptions.Rotate180));

}
```

Scaling an Image

Scaling helps in resizing (reducing and enlarging the size) of an image by changing the number of pixels in the image. Bitmap provides Scaler class for scaling an image. The **Scaler** class requires three properties to scale an image, which are as follows:

- DestinationWidth: The destination width.
- DestinationHeight: The destination height.
- InterpolationMode: The interpolation mode to use when scaling.

The GIF given below shows scaling in and out of an image.



The following code implements scaling in on a button click event and scaling out on another button's click event. The example uses the sample created in Quick Start.

Visual Basic

```
Private Async Function UpdateImageSource() As Task
    Dim sb As SoftwareBitmap = btmp.ToSoftwareBitmap()
    sbs = New SoftwareBitmapSource()
    Await sbs.SetBitmapAsync(sb)
    img.Source = sbs

    img.Width = btmp.PixelWidth
    img.Height = btmp.PixelHeight
End Function
```

```
Private Async Function ApplyTransform(t As BaseTransform) As Task
    Dim bm = btmp.Transform(t)
    btmp.Dispose()
    btmp = bm
    Await UpdateImageSource()
End Function

Private Async Sub btnScale_Click(sender As Object, e As RoutedEventArgs)
    Dim px As Integer = btmp.PixelWidth * 1.6F + 0.5F
    Dim py As Integer = btmp.PixelHeight * 1.6F + 0.5F
    Await ApplyTransform(New Scaler(px, py, InterpolationMode.HighQualityCubic))
End Sub


Private Async Sub btnScaleOut_Click(sender As Object, e As RoutedEventArgs)
    Dim px As Integer = btmp.PixelWidth * 0.625F + 0.5F
    Dim py As Integer = btmp.PixelHeight * 0.625F + 0.5F
    If px > 0 AndAlso py > 0 Then
        Await ApplyTransform(New Scaler(px, py, InterpolationMode.HighQualityCubic))
    End If
End Sub
```

C#

```
private async Task UpdateImageSource()
{
    SoftwareBitmap sb = btmp.ToSoftwareBitmap();
    sbs = new SoftwareBitmapSource();
    await sbs.SetBitmapAsync(sb);
    img.Source = sbs;

    img.Width = btmp.PixelWidth;
    img.Height = btmp.PixelHeight;
}

private async Task ApplyTransform(BaseTransform t)
{
    var bm = btmp.Transform(t);
    btmp.Dispose();
    btmp = bm;
    await UpdateImageSource();
}

private async void btnScale_Click(object sender, RoutedEventArgs e)
{
    int px = (int)(btmp.PixelWidth * 1.6f + 0.5f);
    int py = (int)(btmp.PixelHeight * 1.6f + 0.5f);
    await ApplyTransform(new Scaler(px, py, InterpolationMode.HighQualityCubic));
}

private async void btnScaleOut_Click(object sender, RoutedEventArgs e)
{
    int px = (int)(btmp.PixelWidth * 0.625f + 0.5f);
    int py = (int)(btmp.PixelHeight * 0.625f + 0.5f);
    if (px > 0 && py > 0)
    {
        await ApplyTransform(new Scaler(px, py,
InterpolationMode.HighQualityCubic));
    }
}
```

Working with Bitmap

Working with Bitmap section assumes that you are familiar with the basics and 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 Direct2D Effects
: Learn how to apply Direct2D effects in code.

Applying Direct2D Effects

Direct2D is a 2D 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 the code, Bitmap is converted to a Direct2D bitmap. Direct2D is then used to manipulate the image by applying the built-in effect, shadow, using interoperation with Direct3D API. After all the manipulations, the image is loaded back from Direct2D bitmap to C1Bitmap.

To apply the 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 the following namespaces.

Visual Basic

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

C#

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

2. Create the following class objects.

Visual Basic

```
Private imgSource As SurfaceImageSource
Private sisNative As DXGI.ISurfaceImageSourceNative
Private btmp As C1Bitmap

' device-independent resources
Private d2dF As D2D.Factory2
Private dwF As DW.Factory

' device resources
Private dxgiD As DXGI.Device
Private d2dC As D2D.DeviceContext1

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

C#

```
SurfaceImageSource imgSource;
DXGI.ISurfaceImageSourceNative sisNative;
C1Bitmap btmp;

// device-independent resources
D2D.Factory2 d2dF;
DW.Factory dwF;

// device resources
DXGI.Device dxgiD;
D2D.DeviceContext1 d2dC;

// Direct2D built-in effects
D2D.Effects.Shadow shadow;
D2D.Effects.AffineTransform2D affineT;
D2D.Effects.Composite compst;
```

3. Declare the following constant integers and enumeration.

Visual Basic

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

Private Enum ImageEffect
    Original
    Shadow
End Enum
```

C#

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

enum ImageEffect
{
    Original,
    Shadow
}
```

4. Load the image in C1Bitmap using stream. For details, see Quick Start.

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

Visual Basic

```
' create Direct2D and DirectWrite factories
d2dF = D2D.Factory2.Create(D2D.FactoryType.SingleThreaded)
dwF = DW.Factory.Create(DW.FactoryType.[Shared])

' create GPU resources
CreateDeviceResources()

' create the image source
imgSource = New SurfaceImageSource(marginLT + btmp.PixelWidth + marginRB,
            marginLT + btmp.PixelHeight + marginRB, False)

' obtain the native interface for the image source
sisNative = ComObject.QueryInterface _
            (Of DXGI.ISurfaceImageSourceNative)(imgSource)
sisNative.SetDevice(dxgiD)

' draw the image to SurfaceImageSource
UpdateImageSource(ImageEffect.Original)

' associate the image source with the Image
img.Source = imgSource
```

C#

```
// create Direct2D and DirectWrite factories
d2dF = D2D.Factory2.Create(D2D.FactoryType.SingleThreaded);
dwF = DW.Factory.Create(DW.FactoryType.Shared);

// create GPU resources
CreateDeviceResources();

Unloaded += MainPage_Unloaded;

// create the image source
imgSource = new SurfaceImageSource(marginLT + btmp.PixelWidth + marginRB,
            marginLT + btmp.PixelHeight + marginRB, false);

// obtain the native interface for the image source
sisNative = ComObject.QueryInterface<DXGI.ISurfaceImageSourceNative>
(imgSource);
sisNative.SetDevice(dxgiD);

// draw the image to SurfaceImageSource
UpdateImageSource(ImageEffect.Original);

// associate the image source with the Image
img.Source = imgSource;
```

6. Add the following code to apply the 2D shadow effect.

Visual Basic

```
Private Sub btnShadow_Click(sender As Object, e As RoutedEventArgs) _
Handles btnShadow.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)
    Next
    result.CheckError()
    d3dContext.Dispose()

    'store the DXGI device (for trimming
    'when the Application Is being suspended)
    dxgiD = d3dDevice.QueryInterface(Of DXGI.Device)()
    d3dDevice.Dispose()

    ' create a RenderTarget (DeviceContext for Direct2D drawing)
    Dim d2dDevice = D2D.Device1.Create(d2dF, dxgiD)
    Dim rt = D2D.DeviceContext1.Create(d2dDevice,
             D2D.DeviceContextOptions.None)
    d2dDevice.Dispose()
    rt.SetUnitMode(D2D.UnitMode.Pixels)
    d2dC = rt

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

Private Sub DiscardDeviceResources()
    shadow.Dispose()
    affineT.Dispose()
    compst.Dispose()

    dxgiD.Dispose()
    d2dC.Dispose()
End Sub

Private Sub UpdateImageSource(imageEffect__1 As ImageEffect)
    Dim w As Integer = btmp.PixelWidth + marginLT + marginRB
    Dim h As Integer = btmp.PixelHeight + marginLT + marginRB
    Dim surfaceOffset As Point2L = Point2L.Empty
    Dim dxgiSurface As DXGI.Surface = Nothing
    Dim hr = HResult.Ok

    ' receive the target DXGI.Surface and offset for drawing
    For i As Integer = 0 To 1
        hr = sisNative.BeginDraw(New RectL(w, h),
             surfaceOffset, dxgiSurface)
        If (hr <> DXGI.ResultCode.DeviceRemoved _
            AndAlso hr <> DXGI.ResultCode.DeviceReset) _
            OrElse i > 0 Then
            Exit For
        End If
```

```
            ' try to recreate the device resources
            ' if the old GPU device was removed
            DiscardDeviceResources()
            CreateDeviceResources()
            sisNative.SetDevice(dxgiD)
        Next
        hr.CheckError()

        ' the render target object
        Dim rt = d2dC

        ' create the target Direct2D bitmap for the given DXGI.Surface
        Dim bpTarget = New D2D.BitmapProperties1(New D2D.PixelFormat(
                       DXGI.Format.B8G8R8A8_UNorm, D2D.AlphaMode.Premultiplied),
                       CSng(btmp.DpiX), CSng(btmp.DpiY),
                       D2D.BitmapOptions.Target Or D2D.BitmapOptions.CannotDraw)

        Dim targetBmp = D2D.Bitmap1.Create(rt, dxgiSurface, bpTarget)
        dxgiSurface.Dispose()

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

        ' start drawing
        rt.BeginDraw()

        ' clear the target bitmap
        rt.Clear(Nothing)

        ' convert C1Bitmap image to Direct2D image
        Dim d2dBitmap = btmp.ToD2DBitmap1(rt, D2D.BitmapOptions.None)
        surfaceOffset.Offset(marginLT, marginLT)

        ' apply the effect
        Select Case imageEffect__1
            Case ImageEffect.Original
                rt.DrawImage(d2dBitmap, surfaceOffset.ToPoint2F())
                Exit Select
            Case ImageEffect.Shadow
                rt.DrawImage(ApplyShadow(d2dBitmap), surfaceOffset.ToPoint2F())
                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)

        ' complete drawing on SurfaceImageSource
        sisNative.EndDraw()
    End Sub

    Private Function ApplyShadow(bitmap As D2D.Bitmap1) As D2D.Effect
        shadow.SetInput(0, bitmap)
        shadow.BlurStandardDeviation = 5.0F
        affineT.SetInputEffect(0, shadow)
        affineT.TransformMatrix = Matrix3x2.Translation(20.0F, 20.0F)
        compst.SetInputEffect(0, affineT)
```

```
    compst.SetInput(1, bitmap)
    Return compst
End Function

Private Sub MainPage_Unloaded(sender As Object, e As RoutedEventArgs)
    DiscardDeviceResources()

    btmp.Dispose()
    d2dF.Dispose()
    dwF.Dispose()

    img.Source = Nothing
    sisNative.Dispose()
    sisNative = Nothing
End Sub
```

C#

```
private void btnShadow_Click(object sender, RoutedEventArgs 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)
    dxgiD = d3dDevice.QueryInterface<DXGI.Device>();
    d3dDevice.Dispose();

    // create a RenderTarget (DeviceContext for Direct2D drawing)
    var d2dDevice = D2D.Device1.Create(d2dF, dxgiD);
    var rt = D2D.DeviceContext1.Create(d2dDevice,
D2D.DeviceContextOptions.None);
    d2dDevice.Dispose();
    rt.SetUnitMode(D2D.UnitMode.Pixels);
    d2dC = rt;

    // create built-in effects
```

```
    shadow = D2D.Effects.Shadow.Create(rt);
    affineT = D2D.Effects.AffineTransform2D.Create(rt);
    compst = D2D.Effects.Composite.Create(rt);
}

void DiscardDeviceResources()
{
    shadow.Dispose();
    affineT.Dispose();
    compst.Dispose();

    dxgiD.Dispose();
    d2dC.Dispose();
}

void UpdateImageSource(ImageEffect imageEffect)
{
    int w = btmp.PixelWidth + marginLT + marginRB;
    int h = btmp.PixelHeight + marginLT + marginRB;
    Point2L surfaceOffset = Point2L.Empty;
    DXGI.Surface dxgiSurface = null;
    var hr = HResult.Ok;

    // receive the target DXGI.Surface and offset for drawing
    for (int i = 0; i <= 1; i++)
    {
        hr = sisNative.BeginDraw(new RectL(w, h),
             out surfaceOffset, out dxgiSurface);
        if ((hr != DXGI.ResultCode.DeviceRemoved &&
             hr != DXGI.ResultCode.DeviceReset) || i > 0)
        {
            break;
        }

        // try to recreate the device resources
        // if the old GPU device was removed
        DiscardDeviceResources();
        CreateDeviceResources();
        sisNative.SetDevice(dxgiD);
    }
    hr.CheckError();

    // the render target object
    var rt = d2dC;

    // create the target Direct2D bitmap for the given DXGI.Surface
    var bpTarget = new D2D.BitmapProperties1( new D2D.PixelFormat(
                   DXGI.Format.B8G8R8A8_UNorm, D2D.AlphaMode.Premultiplied),
                   (float)btmp.DpiX, (float)btmp.DpiY,
                   D2D.BitmapOptions.Target | D2D.BitmapOptions.CannotDraw);

    var targetBmp = D2D.Bitmap1.Create(rt, dxgiSurface, bpTarget);
    dxgiSurface.Dispose();

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

    // start drawing
    rt.BeginDraw();

    // clear the target bitmap
```

```
        rt.Clear(null);

        // convert C1Bitmap image to Direct2D image
        var d2dBitmap = btmp.ToD2DBitmap1(rt, D2D.BitmapOptions.None);
        surfaceOffset.Offset(marginLT, marginLT);

        // apply the effect
        switch (imageEffect)
        {
            case ImageEffect.Original:
                rt.DrawImage(d2dBitmap, surfaceOffset.ToPoint2F());
                break;
            case ImageEffect.Shadow:
                rt.DrawImage(ApplyShadow(d2dBitmap), surfaceOffset.ToPoint2F());
                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);

        // complete drawing on SurfaceImageSource
        sisNative.EndDraw();
    }

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

    private void MainPage_Unloaded(object sender, RoutedEventArgs e)
    {
        DiscardDeviceResources();

        btmp.Dispose();
        d2dF.Dispose();
        dwF.Dispose();

        img.Source = null;
        sisNative.Dispose();
        sisNative = null;
    }
```

Bitmap Samples

With the C1Studio installer, you get Bitmap samples that help you understand the implementation of the product. The C# and VB samples are available at the default installation folder - Documents\ComponentOne Samples\UWP\C1.UWP.Bitmap.

The C# samples available at the default installation location is as follows:

Sample	Description
BitmapSamples	This sample shows how to display an animated gif using BitmapImage, use C1Bitmap to crop an image, use C1Bitmap and vertex shader to distort an image, and to use various transformation, such as cropping, scaling, rotation, and flipping on a bitmap image.
Direct2DEffects	This sample loads an image in C1Bitmap, converts it to Direct2D bitmap, applies various effects and draws to SurfaceImageSource. When the user clicks the Export button the image is converted to Direct2D bitmap, then used as the source for a Direct2D effect. The result is imported into another instance of C1Bitmap, then stored to a file.

The VB samples available at the default installation location is as follows:

Sample	Description
BitmapSamples	This sample shows how to display an animated gif using BitmapImage, use C1Bitmap to crop an image, use C1Bitmap and vertex shader to distort an image, and to use various transformation, such as cropping, scaling, rotation, and flipping on a bitmap image.
Direct2DEffects	This sample loads an image in C1Bitmap, converts it to Direct2D bitmap, applies various effects and draws to SurfaceImageSource. When the user clicks the Export button the image is converted to Direct2D bitmap, then used as the source for a Direct2D effect. The result is imported into another instance of C1Bitmap, then stored to a file.