Bitmap for UWP

2018.04.10 更新

グレープシティ株式会社

Bitmap for UWP

目次

Bitmap for UWP

ComponentOne には、画像を読み込み、保存、変換するためのクラスライブラリである **Bitmap for UWP** が導入されています。Bitmap を使用すると、イメージファイル上でクリップ、反転、拡大/縮小、回転、またはこれらの変換の任意の組み合わせを適用することができます。さらに、Bitmap ではさまざまな画像処理ニーズに対応するために、BMP、PNG、JPG などのさまざまなコンテナ形式をサポートしており、画像のピクセル形式を変更することができます。

主な特長

Bitmap は画像処理のためにサポートされる機能のいくつかを提供します。シンプルな画像の表示と保存だけでなく、以下に示すさまざまな目的のために Bitmap を使用することができます。

- **画像の読み込み**
  Bitmap は、BMP、PNG、JPEG、JPEG-XR、ICO などのさまざまなコンテナ形式の画像を読み込みます。ビットマップは単一フレームの TIFF と GIF もサポートしています。画像は、ファイルやストリームから読み込んだり、他のビットマップオブジェクトを参照したり、配列から取得することができます。さらに、Bitmap では **C1Bitmap** の同一インスタンス内に複数の画像をを1つずつ読み込むことができます。

- **画像の保存**
  読み込み時と同様に、C1Bitmap に読み込まれた画像は、StorageFile、**IOutputStream**、または **System.IO.Stream** に保存することができます。C1Bitmap は、BMP、PNG、JPEG、JPEG-XR、TIFF、GIF（単一フレーム）など、サポートされているコンテナ形式ごとに特定の **SaveAs** メソッドを提供します。

- **画像の変換**
  Bitmap を使用して、画像にさまざまな変換を適用できます。例えば、**Transform** メソッドを適用して画像を簡単にクリップ、クロップ、回転、拡大/縮小することができます。

- **Direct2D エフェクトの適用**
  Bitmap では、画像に対して Direct2D エフェクトを適用し、さまざまなアニメーションやイメージングエフェクトを作成できます。

- **プラットフォーム固有のビットマップオブジェクトとの相互変換**
  Bitmap の重要な機能の1つとして、プラットフォーム固有のビットマップオブジェクトとの相互変換をサポートします。例えば、Windows Store アプリケーションにおいて、**C1Bitmap** を **WriteableBitmap** または **SoftwareBitmap** に簡単に変換できます。また、例えば既存の **C1Bitmap** のフラグメントとして **WriteableBitmap** をインポートしたり、**SoftwareBitmap** のインスタンスから新規に **C1Bitmap** を作成することができます。

オブジェクトモデルの概要

Bitmap には、さまざまなクラス、オブジェクト、コレクション、および関連する画像処理用のメソッドおよびプロパティを提供する、リッチなオブジェクトモデルが付属しています。以下の表は、これらのオブジェクトの一部とその主要なプロパティを示します。

C1Bitmap
プロパティ:ContainerFormat、HasImage、HasMetadata、ImagingFactory、IsDisposed、NativeBitmap、PixelFormat、PixelHeight、PixelWidth **メソッド: Dispose、Import、Load、LoadAsync、LoadMetadata、Save、SaveAsBmp、SaveAsGif、SaveAsJpeg、SaveAsPng、SaveAsTiff、SaveAsync、ToSoftwareBitmap、ToWritableBitmap、Transform**
Clipper
プロパティ:ImageRect
FlipRotator
プロパティ:TransformOptions
FormatConverter
プロパティ:DestinationFormat、Palette、PaletteTranslate
Scaler
プロパティ:DestinationHeight、DestinationWidth、InterpolationMode

クイックスタート

このクイックスタートでは、**Bitmap**を使用して画像を読み込む方法を説明します。Visual Studio で UWP アプリケーションを作成し、C1.UWP.Bitmap (dll) への参照、イメージコントロール、およびストリームから **C1Bitmap** に画像を読み込むためのボタンを追加することから始めます。

C1Bitmap オブジェクトに画像を読み込むシンプルな UWP アプリケーションを作成するには、次の手順を実行してください。

1. **アプリケーションの設定**
2. **C1Bitmap 内に画像を読み込む**

以下の画像は、ストリームから **C1Bitmap** 内に読み込まれた画像を表示する例を示しています。

画像の読み込み

アプリケーションの設定

アプリケーションを設定するには、以下の手順に従ってください。

1. **Visual Studio** で新規プロジェクトを作成し、**空白のアプリ(ユニバーサルWindows)**を選択します。
2. アプリケーションに以下の参照を追加します。
    - C1.UWP.Bitmap
    - C1.UWP.DX
3. 「**ソリューションエクスプローラ**」内で、プロジェクト名を右クリックして、[**追加**]→[**新しいフォルダ**]を選択し、追加されたフォルダに新しい名前を付けます。この例では、新規フォルダに「**Resources**」という名前を付けています。
4. サンプル画像を Resources フォルダに追加し、「**プロパティ**」ウィンドウにてその「**ビルド アクション**」プロパティを「**埋め込みリソース**」に設定します。
5. サンプル画像を読み込むための標準の **Button** コントロールと、サンプル画像を表示するための img と名前付けされた **Image** コントロールを追加します。
6. ボタンの **Content** プロパティに「**画像の読み込み**」というテキストを設定します。

先頭に戻る

C1Bitmap 内に画像を読み込む

C1Bitmap 内に画像を読み込むには、以下の手順に従ってください。

1. コードビューに切り替えて、コード内に以下の import ステートメントを追加します。
    - **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. 以下のクラスオブジェクトを作成します。
   - **Visual Basic**

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

   - **C#**

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

3. ビットマップと SoftwareBitmapSource を初期化するために、クラスのコンストラクタに以下のコードを追加します。
   - **Visual Basic**

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

   - **C#**

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

4. 以下のコードを **btnLoad_Click** イベント内に追加します。これは、ストリームを使用して **C1Bitmap** 内に画像を読み込みます。
   - **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();
```

> **メモ:** イベント上で非同期メソッドを呼び出すため、async キーワードと await オペレータを使用しています。

5. **UpdateImageSource** という名前のメソッドを作成します。これは、**SoftwareBitmap** を作成し、それをソースの SoftwareBitmap としてイメージソースに割り当てます。
   - **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;
}
```

先頭に戻る

機能

このセクションには、Bitmap コントロールで使用できるすべての機能が含まれています。

画像の読み込みおよび保存
　読み込みおよび保存をコード内に実装する方法を学びます。
変換の適用
　さまざまな変換をコード内に適用する方法を学びます。

画像の読み込みおよび保存

Bitmap では、**C1Bitmap** クラスの **Load** メソッドを使用して、**C1Bitmap** オブジェクト内に画像を読み込むことができます。**Load** および **LoadAsync** のオーバーロードメソッドを使用すると、画像を StorageFile、IInputStream、または System.IO.Stream から読み込むことができます。**Bitmap** では、画像のサイズ、解像度（DPI）、またはピクセル形式を決定するために使用できる画像メタデータを読み込むこともできます。さらに、いくつかの画像を **C1Bitmap** の同じインスタンスに1つずつ読み込みまたはインポートすることができます。

読み込みと同様に、**C1Bitmap** の画像は、StorageFile、IOutputStream、あるいは System.IO.Stream に保存することができます。ビットマップは、コンテナ形式を引数として受け入れる **C1Bitmap** クラスの一般的な **Save** メソッドを提供します。また、サポートされている各コンテナ形式に対して個別の **SaveAs** メソッドを提供します。

ここでは、ファイルから任意の画像を読み込む方法について説明します。ストリームから画像を読み込む方法については「クイックスタート」を参照してください。

以下の手順は、ファイルから任意の画像を読み込む方法を示しています。このコードは、**LoadAsync** メソッドを使用して StorageFile から画像を読み込みます。

1. 以下のクラスオブジェクトを作成して初期化します。
   - **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. StorageFile から画像を読み込むために、以下のコードを追加します。
   - **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();
}
```

メモ: イベント上で非同期メソッドを呼び出すため、async キーワードと await オペレータを使用しています。

3. **UpdateImageSource** という名前のメソッドを作成します。これは、SoftwareBitmap を作成し、それをソースの SoftwareBitmap としてイメージソースに割り当てます。
   - **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. 次のコードを使用して、読み込まれた任意の画像を保存します。ここでは、**SaveAsPngAsync** メソッドを使用して、画像を PNG 形式の StorageFile に保存します。
   - **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("ファイルが保存されました。")

    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("ファイルが保存されました。");
    await md.ShowAsync();
```

```
    }
    btmp.Dispose();
```

変換の適用

Bitmap は、クリッピング、反転、拡大/縮小、回転など、画像に対してさまざまな変形を適用できます。これらの変換とその実装方法について学びます。

画像のクリッピング
: コード内にクリッピング処理を実装する方法を学びます。

画像の反転
: コード内に反転処理を実装する方法を学びます。

画像の回転
: コード内に回転処理を実装する方法を学びます。

画像の拡大/縮小
: コード内に拡大/縮小処理を実装する方法を学びます。

画像のクリッピング

クリッピングとは、画像の一部を切り抜くことです。ビットマップでは、**Clipper** クラスで画像を切り取ることができます。画像全体ではなく、ソース画像をクリップして小さなフラグメントを読み込むため、**Clipper** クラスを使用してクリッパー変換を渡すことができます。

以下のアニメーションは画像のクリッピングを示しています。

以下のコードは、ボタンのクリックイベントによる画像の切り抜きを実装しています。この例では、「 クイックスタート」で作成したサンプルを使用します。

- **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) });
}
```

画像の反転

画像の反転とは、垂直あるいは水平方向に元画像の鏡像を作成することです。Bitmap では、**FlipRotator** クラスの **TransformOptions** プロパティを使用して、画像を水平方向あるいは垂直方向に反転させることができます。**TransformOptions** プロパティは、**TransformOptions** 列挙値にて変換オプションを設定します。

以下の画像は、水平方向に反転した画像を示しています。

画像の読み込み　画像の反転

以下のコードは、ボタンのクリックイベントでイメージを水平方向に反転させる方法を実装しています。この例では、「クイックスタート」セクションで作成したサンプルを使用します。

- **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));
}
```

同様に、**TransformOptions** 列挙型の **FlipVertical** 値を使用して画像を垂直方向に反転できます。

画像の回転

Bitmap は、画像を時計回りに90度、180度、270度の異なる角度に回転させる柔軟性を提供します。**C1Bitmap** によって提供される **FlipRotator** クラスの **TransformOptions** プロパティを使用して画像を回転することができます。**TransformOptions** プロパティは、変換オプションを設定するために **TransformOptions** 列挙値を受け入れます。

以下の画像は、時計回りに180度回転した画像を示しています。

画像の読み込み　画像の回転

以下のコードは、ボタンのクリックイベントにて画像を時計回りに180度回転する処理を実装します。この例では、「クイックスタート」セクションで作成したサンプルを使用します。

- **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));

}
```

画像の拡大/縮小

スケーリングとは、画像のピクセル数を変更することによって、画像のサイズ変更（サイズの拡大/縮小）を行います。Bitmap には、画像を拡大/縮小するための **スケーラ** クラスが用意されています。**Scaler** クラスでは、画像を拡大/縮小するために次の3つのプロパティが必要です。

- **DestinationWidth**: 目的の幅です。
- **DestinationHeight**: 目的の高さです。
- **InterpolationMode**: スケーリング時に使用する補完モードです。

以下のアニメーションは画像の拡大/縮小を示しています。

以下のコードは、ボタンのクリックイベントによる画像の拡大および縮小を実装しています。この例では、「クイックスタート」で作成したサンプルを使用します。

- **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));
    }
}
```

Bitmap の操作

「Bitmap の操作」セクションは、ユーザーの皆様が Bitmap コントロールの基礎と機能および一般的な使用方法を理解していることを前提としています。次のセクションでは、Bitmap で提供されている補助機能について説明します。

Direct2D エフェクトの適用
　Direct2D エフェクトをコードで適用する方法を説明します。

Direct2D エフェクトの適用

Direct2D は、Microsoft によって設計された 2D グラフィック API で、画像を操作するための広範な組み込みおよびカスタムのエフェクトが提供されています。この API を使用すると、ビットマップ、2D ジオメトリ、テキストの高品質で高速なレンダリングが可能です。

Bitmap では、Direct2D のエフェクトを使用したり、画像にエフェクトを適用することができます。Bitmap を使用して適用できる画像エフェクトを次に一覧します。

- ブラー（ガウス）
- シャープネス
- 水平スミア
- シャドウ
- ディスプレイスメントマップ
- エンボス
- エッジ検出
- セピア

これらのエフェクトから 1 つを選んで画像に適用してみましょう。次の図は、Bitmap で Direct2D を使用する例として、組み込み 2D エフェクトの 1 つ、シャドウを示しています。

コードで、Bitmap が Direct2D ビットマップに変換されます。次に Direct2D を使用して画像を操作し、Direct3D API との相互運用によって組み込みエフェクト、シャドウが適用されます。すべての操作が完了したら、画像が Direct2D ビットマップから C1Bitmap にロードし直されます。

画像にシャドウエフェクトを適用するには、**C1.Util.DX.Direct2D.Effects** 名前空間のメンバクラスである **Shadow**、**AffineTransform2D**、**Composite** のプロパティを使用します。

以下の手順は、2D シャドウエフェクトを画像に適用する方法を示します。この例では、「クイックスタート」で作成したサンプルを使用します。

1. 次の名前空間を追加します。
   - **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. 次のクラスオブジェクトを作成します。
   - **Visual Basic**

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

' 装置独立リソース
Private d2dF As D2D.Factory2
Private dwF As DW.Factory

' 装置リソース
Private dxgiD As DXGI.Device
Private d2dC As D2D.DeviceContext1

' Direct2Dの組み込み効果
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;

// 装置独立リソース
D2D.Factory2 d2dF;
DW.Factory dwF;

// 装置リソース
DXGI.Device dxgiD;
D2D.DeviceContext1 d2dC;

// Direct2Dの組み込み効果
D2D.Effects.Shadow shadow;
D2D.Effects.AffineTransform2D affineT;
D2D.Effects.Composite compst;
```

3. 次の整数定数と列挙を宣言します。
   - **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. ストリームを使用して画像を C1Bitmap にロードします。詳細については、「クイックスタート」を参照してください。
5. 次のコードを追加して、リソースと画像ソースを作成し、画像ソースを画像と関連付けます。
   - **Visual Basic**

```
'  Direct2DおよびDirectWriteファクトリを作成します
d2dF = D2D.Factory2.Create(D2D.FactoryType.SingleThreaded)
dwF = DW.Factory.Create(DW.FactoryType.[Shared])

' GPUリソースを作成します
CreateDeviceResources()

' イメージソースを作成します
imgSource = New SurfaceImageSource(marginLT + btmp.PixelWidth + marginRB,
            marginLT + btmp.PixelHeight + marginRB, False)

' イメージソースのネイティブインタフェースを取得します
sisNative = ComObject.QueryInterface _
            (Of DXGI.ISurfaceImageSourceNative)(imgSource)
sisNative.SetDevice(dxgiD)

' SurfaceImageSourceに画像を描画します
UpdateImageSource(ImageEffect.Original)

' イメージソースをイメージに関連付けます
img.Source = imgSource
```

   - **C#**

```
//  Direct2DおよびDirectWriteファクトリを作成します
d2dF = D2D.Factory2.Create(D2D.FactoryType.SingleThreaded);
dwF = DW.Factory.Create(DW.FactoryType.Shared);

// GPUリソースを作成します
CreateDeviceResources();

Unloaded += MainPage_Unloaded;

// イメージソースを作成します
imgSource = new SurfaceImageSource(marginLT + btmp.PixelWidth + marginRB,
            marginLT + btmp.PixelHeight + marginRB, false);

// イメージソースのネイティブインタフェースを取得します
sisNative = ComObject.QueryInterface<DXGI.ISurfaceImageSourceNative>(imgSource);
sisNative.SetDevice(dxgiD);

// SurfaceImageSourceに画像を描画します
UpdateImageSource(ImageEffect.Original);

// イメージソースをイメージに関連付けます
img.Source = imgSource;
```

6. 次のコードを追加して、2D シャドウエフェクトを適用します。

- **Visual Basic**

```
Private Sub btnShadow_Click(sender As Object, e As RoutedEventArgs) _
Handles btnShadow.Click
    UpdateImageSource(ImageEffect.Shadow)
End Sub

Private Sub CreateDeviceResources()
    ' Direct3Dデバイスを作成します
    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
        ' ハードウェアが利用できない場合はWARPを使用します
        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()

    ' DXGI装置を格納します(アプリケーションが
    ' 中断されているときにトリミングするため)
    dxgiD = d3dDevice.QueryInterface(Of DXGI.Device)()
    d3dDevice.Dispose()

    ' RenderTargetを作成します(Direct2D描画用のDeviceContext)
    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

    ' 組み込みの効果を作成します
    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

    ' 描画のためにDXGI.Surfaceとオフセットを受け取ります
    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

            ' 古いGPUデバイスが削除された場合は、
            ' 装置リソースを再作成してみてください
            DiscardDeviceResources()
            CreateDeviceResources()
            sisNative.SetDevice(dxgiD)
        Next
        hr.CheckError()

        ' レンダー対象オブジェクト
        Dim rt = d2dC

        ' 対象Direct2Dビットマップを作成します
        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()

        ' 対象ビットマップをレンダー対象に関連付けます
        rt.SetTarget(targetBmp)
        targetBmp.Dispose()

        ' 描画を開始します
        rt.BeginDraw()

        ' 対象ビットマップをクリアします
        rt.Clear(Nothing)

        ' C1Bitmap画像をDirect2D画像に変換します
        Dim d2dBitmap = btmp.ToD2DBitmap1(rt, D2D.BitmapOptions.None)
        surfaceOffset.Offset(marginLT, marginLT)

        ' 効果を適用します
        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()

        ' 描画を終了します(すべての描画コマンドはその時点で実行されます)
        rt.EndDraw()

        ' 対象ビットマップをデタッチして破棄します
        rt.SetTarget(Nothing)

        ' SurfaceImageSourceでの完全な描画
        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()
{
    // Direct3Dデバイスを作成します
    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++)
    {
        // ハードウェアが利用できない場合はWARPを使用します
        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();

    // DXGI装置を格納します(アプリケーションが
    // 中断されているときにトリミングするため)
    dxgiD = d3dDevice.QueryInterface<DXGI.Device>();
    d3dDevice.Dispose();

    // RenderTargetを作成します(Direct2D描画用のDeviceContext)
    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;

    // 組み込みの効果を作成します
```

```
        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;

        // 描画のためにDXGI.Surfaceとオフセットを受け取ります
        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;
            }

            // 古いGPUデバイスが削除された場合は、
            // 装置リソースを再作成してみてください
            DiscardDeviceResources();
            CreateDeviceResources();
            sisNative.SetDevice(dxgiD);
        }
        hr.CheckError();

        // レンダー対象オブジェクト
        var rt = d2dC;

        // 対象Direct2Dビットマップを作成します
        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();

        // 対象ビットマップをレンダー対象に関連付けます
        rt.SetTarget(targetBmp);
        targetBmp.Dispose();

        // 描画を開始します
        rt.BeginDraw();

        // 対象ビットマップをクリアします
        rt.Clear(null);
```

```
    // C1Bitmap画像をDirect2D画像に変換します
    var d2dBitmap = btmp.ToD2DBitmap1(rt, D2D.BitmapOptions.None);
    surfaceOffset.Offset(marginLT, marginLT);

    // 効果を適用します
    switch (imageEffect)
    {
        case ImageEffect.Original:
            rt.DrawImage(d2dBitmap, surfaceOffset.ToPoint2F());
            break;
        case ImageEffect.Shadow:
            rt.DrawImage(ApplyShadow(d2dBitmap), surfaceOffset.ToPoint2F());
            break;
    }

    d2dBitmap.Dispose();

    // 描画を終了します（すべての描画コマンドはその時点で実行されます）
    rt.EndDraw();

    // 対象ビットマップをデタッチして破棄します
    rt.SetTarget(null);

    // SurfaceImageSourceでの完全な描画
    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;
}
```