Reviewer's Guide

GrapeCity Documents for Imaging, .NET Edition
(CTP)

GRAPECITY INC.

IMPORTANT NOTICE

THIS CTP VERSION WILL EXPIRE ON MAR 31, 2019
AND IS NOT LICENSED FOR ANY TYPE OF DISTRIBUTION

Table of Contents

Welcome to the GcImaging Beta Test

Thank you for taking the opportunity to review GrapeCity Documents for Imaging, .NET Edition (GcImaging) (CTP). This guide will provide you a quick way of learning the primary features and a tutorial to get started with the product.

What we want from our testers?

Help us evaluate GcImaging and let us know –

1. The use cases or missing features that cannot be fulfilled using GcImaging in your applications.
2. The use cases where GcImaging can be successfully used in your applications.
3. Any other suggestions for the product.

Where to send feedback

- Email your responses to gcdocsbeta@grapecity.com
- Post in the forum at https://www.grapecity.com/en/forums/documents-imaging

GcImaging Feature List

GcImaging is an image library that can create, load, modify and save images in .NET Standard 2.0 applications with full support on Windows, MAC and Linux. The library is a feature-rich API that can load images of formats like JPEG, PNG, TIFF, GIF and BMP apply advanced effects and save them back. You can read/write images, modify images like rotate, crop, resize, convert images, apply dithering, thresholding effects on grayscale images, apply effects on RGB images, draw and fill graphics on images, draw advanced text with full font handling and text, paragraph formatting on images, apply advanced TIFF features etc.

- **Platform supported**
  - .NET Standard 2.0/.NET Core 2.1 (full support on Windows, Linux and MacOS)

- **Read/Write Images**
  - Read from File, Stream or Byte Array
  - Write image to File or Stream
  - Supported image formats
    - BMP
    - JPEG
    - GIF (single frame only)
    - PNG
    - TIFF

- **Process Images**
  - Rotate and Flip images
  - Crop images
  - Resize images with several interpolation modes
    - Nearest neighbor interpolation algorithm
    - Bilinear interpolation algorithm
    - Bicubic interpolation algorithm
    - Resampling algorithm for downscaling
  - Convert images to Grayscale (monochromatic gray) in accordance with standards
    - BT.601 (old standard for color TV and video systems such as PAL and NTSC)
    - ITU-R BT.709 standard used for HDTV
    - ITU-R BT.2100 standard for HDR television
  - Apply Dithering effects to Grayscale and RGB images (to reduce the number of bits per channel)
    - Floyd-Steinberg dithering
    - Jarvis, Judice, and Ninke filter
    - Stucki dithering
    - Bill Atkinson's formula
    - Burkes dithering
    - Frankie Sierra's dithering
    - Simplified two-row Sierra algorithm
    - Sierra Lite dithering
  - Apply Thresholding effects to Grayscale images
    - Bradley and Roth's method of adaptive image thresholding

        - Otsu's method of clustering-based image thresholding
    - Apply more effects to RGB images
        - HueRotationEffect
        - SaturationEffect
        - SepiaEffect
        - TemperatureAndTintEffect
        - LuminanceToAlphaEffect
        - OpacityEffect
    - Apply a 5x4 color matrix to an image
    - Apply a bi-level or half-tone transparency mask to an image
    - Set the background color for a semi-transparent image (make the image opaque)
    - Check whether an image has transparent pixels
    - Get the list of unique pixel colors in the image
    - Convert full-colored images to indexed images with palette
    - Change Image resolution
    - Clear image

- **Draw graphics on Images**
    - Draw and fill geometric figures
        - Lines
        - Polygons
        - Rectangles
        - Rounded Rectangles
        - Ellipses
        - Paths
    - Create paths consisting of:
        - Lines
        - Cubic Bezier curves
        - Quadratic Bezier curves
        - Elliptical arcs
    - Draw with pens using various dash styles, line caps, and line joins
    - Fill with solid and gradient brushes
    - Draw images with various alignment, tiling, opacity, and interpolation modes
    - Clipping all drawing operations to a rectangular region
    - Clipping drawing operations to a region defined as an arbitrary path
    - Apply matrix transformations to drawing operations
    - Aliased and anti-aliasing modes for rendering text and graphics
    - Create regions composed of figures and paths, use regions for clipping and filling
    - Combine regions with operations like Union, Intersect, Xor, Exclude
    - Draw text and graphics in multi-threaded mode to boost performance
    - Each image can be drawn and processed in a separate thread.

- **Draw text on Images**
    - Powerful text layout engine with full support for TrueType, OpenType, WOFF fonts.
    - Text processing, layout and formatting is platform-independent and follows Unicode standards.
    - Fallback, linked and EUDC fonts are fully supported.
    - Draw colored fonts and characters with 4-byte Unicode codes.
    - Font features and bold/italic emulation are supported.

  - Right-to-left and Far Eastern languages are fully supported, including Kashida in Arabic and vertical text.
  - Various paragraph formatting options, support for multi-column text with balanced columns.
  - Measure and split text.
  - Draw text around images or other rectangular areas.

- **TIFF Images**
  - Read and write TIFF frames as separate images
  - Supported compression schemes:
    - Uncompressed
    - CCITT 1D
    - Group 3 Fax
    - Group 4 Fax
    - LZW (with optional Differencing Predictor)
    - Deflate (with optional Differencing Predictor)
    - PackBits (RLE)
  - Supported PhotometricInterpretation tag values:
    - WhiteIsZero
    - BlackIsZero
    - RGB (with Chunky and Planar format)
    - RGB Palette (16-color and 256-color palettes only)
    - Transparency mask
  - Support for tiled Images
  - Various number of bits per channel (from 1 to 16) for ARGB and Grayscale images
  - Support for Associated and Unassociated alpha channel
  - Support metadata for TIFF frames, such as image description, etc.

GcImaging Architecture Overview

The main license-able class is **GrapeCity.Documents.Imaging.** GcBitmap, living in the GrapeCity.Documents.Imaging assembly. GcBitmap does not derive from any public classes.

GcBitmap

Instances of the GcBitmap class can be created directly (new GcBitmap()).

GcBitmapGraphics

An instance of GcBitmapGraphics can be created on a GcBitmap using the method GcBitmap.CreateGraphics(). GcBitmapGraphics derives from GcGraphics and provides drawing, filling, clipping and other normal graphics operations. In particular, text can be drawn on a GcBitmapGraphics using the DrawString() and DrawTextLayout() methods.

BitmapRenderer

BitmapRenderer is a public class that provides access to low level bitmap operations and properties. This includes the basic graphics operations such as drawing and filling, and bitmap specific stuff like antialiasing etc. Most properties and methods in GcBitmapGraphics are implemented via BitmapRenderer, but that class can also be used directly if needed. An instance of BitmapRenderer can be acquired via the GcBitmap.Renderer property.

Drawing a GcBitmap on a GcGraphics

As mentioned above, we already have the class GrapeCity.Documents.Drawing.Image. Instances of that class can be created from files (e.g. .JPEG or .PNG), streams or bytes in memory, and can be drawn on instances of GcGraphics (e.g. onto a PDF page using GcPdfGraphics) using the GcGraphics.DrawImage() method. To provide a bridge between GcBitmap and GrapeCity.Documents.Drawing.Image, a new static method of Image class can be used : Image.FromGcBitmap(). This method takes as argument an instance of GcBitmap and creates an Image that can be then drawn onto a GcGraphics, including GcPdfGraphics and GcBitmapGraphics.

In the GrapeCity.Documents.Drawing.Image there is a method AsGcBitmap(), which allows to acquire a GcBitmap from an instance of the Image class.

GcWicBitmap

GcWicBitmap is a Windows specific class, which represents a bitmap from the Windows Imaging Component (WIC) and is part of GrapeCity.Documents.Common.Windows library. GcWicBitmap provides direct access to WIC-specific properties and methods, which are very different from the object model of GcBitmap.

GcWicBitmap provides the same kind of functionality as GcBitmap, but in the WIC-specific way, using WIC and Direct2D types and interfaces.

As with GcBitmap, there is a class GcWicBitmapGraphics which implements GcGraphics on a GcWicBitmap.

The primary reason for the existence of GcWicBitmap is to allow users on Windows a faster (approx 3 times) and sometimes yielding higher-quality results, way to manipulate images, that is Windows-specific using Direct2D. Some features like support for JpegXR format, reading TIFF-JPEG images or using the ordered dither algorithms are currently available with GcWicBitmap only. If you do not want to write Windows-specific code, you can safely ignore GcWicBitmap and just use GcBitmap.

Getting a GcBitmap from a GcWicBitmap and vice versa

A GcBitmap can be acquired from GcWicBitmap and overload of the GcWicBitmap.ToGcBitmap() method.

To copy a non-opaque image from GcBitmap to GcWicBitmap you have to pre-multiply the color channels by the alpha channel in GcBitmap with the GcBitmap.ConvertToPremultipliedAlpha() method. Then, call the GcWicBitmap.Import() method passing GcBitmap as an argument.

How to use GcImaging in Visual Studio (Windows, MAC, Linux)

Visual Studio 2017 on Windows

1. Create any application (.NET Core, ASP.NET Core, .NET Framework - any target that supports .NET Standard 2.0).
2. Right-click the project in Solution Explorer and choose **Manage NuGet Packages**.
3. In the **Package source** on top right, choose **nuget.org**.
4. Select **Include prerelease** check box.
5. Click **Browse** tab on top left and search for "Grapecity.Documents".
6. On the left panel, choose **GrapeCity.Documents.Imaging.**
7. On the right panel, click **Install**.
8. Choose **Accept** in the next screen.
9. After the installation is complete, go to the **NuGet** folder in the solution explorer and confirm that the required packages are added to your project as shown below.

This will add references of the package to your application. After this step, follow the steps in the Quick Start topic below.

Visual Studio for MAC

1. Create any application (.NET Core, ASP.NET Core, .NET Framework - any target that supports .NET Standard 2.0).
2. In the tree view on the left, right-click **Dependencies** and choose **Add Packages**.
3. Under 'nuget.org', select '**Show pre-release packages**'.
4. In the Search panel, type GrapeCity.
5. From the list of packages displayed in the left panel, select '**GrapeCity.Documents.Imaging'** and click **Add Packages.**
6. Click **Accept**.

This will add references of the package to your application. After this step, follow the steps in the Quick Start topic below.

Visual Studio Code for Linux

1. Open Visual Studio Code.
2. Install **Nuget Package Manager** from Extensions.
3. Create a folder 'MyApp' in your Home folder.
4. In the Terminal in Visual Studio Code, type – cd MyApp
5. Type following command – dotnet new console
6. Observe: This creates a .NET Core application with MyApp.csproj file and Program.cs.
7. Press Ctrl+P. This would open a command line at the top.
8. Type following command – '>'
9. Observe: This will show option '**Nuget Package Manager: Add Package'**.

10. Click the above option.
11. Type – 'GrapeCity' and press Enter.
12. Observe: This would show GrapeCity packages in the dropdown.
13. Choose **GrapeCity.Documents.Imaging.**
14. Observe: The package would be added to your .csproj file.
15. Type following command in the Terminal window:
    ```
    dotnet restore
    ```

This will restore the packages needed in your application.

After this step, follow the steps in the Quick Start topic below.

Quick Start

Let's create a new image with GcImaging in .NET Core application.

1. In the Program.cs file, include following namespaces -

```
using System.IO;

using System.Drawing;

using GrapeCity.Documents.Drawing;

using GrapeCity.Documents.Text;

using GrapeCity.Documents.Imaging;
```

2. Define color object and image parameters.

```
var blue = Color.FromArgb(46, 72, 132));
int pixelWidth = 1024;

int pixelHeight = 1024;

bool opaque = true;

float dpiX = 96;

float dpiY = 96;
```

3. Create Bitmap image object –

```
using (var bmp = new GcBitmap(pixelWidth, pixelHeight, opaque, dpiX, dpiY))
```

4. Create graphics on the image -

```
using (var g = bmp.CreateGraphics(blue))

{

        var rc = new RectangleF(0, 0, pixelWidth, pixelHeight);

        var b = new RadialGradientBrush(Color.White, blue, new PointF(0.5f,
        0.5f), true);

        g.FillRectangle(rc, b);
```

5. Add text on the image

```
var tf = new TextFormat
{
        Font = Font.FromFile(Path.Combine("Resources","Fonts",
        "times.ttf")),
        FontSize = 40
};
g.DrawString("Hello, World!", tf, rc, TextAlignment.Center,
ParagraphAlignment.Center, false);
```

6. Save the image as PNG.

```
        bmp.SaveAsPng("HelloWorld.png");
}
```

7. On Visual Studio for Windows, MAC, just press the 'Start Debugging' button.
   Or,
   On Visual Studio Code for Linux, type following commands in Terminal –

```
cd <Your Project folder>
dotnet run
```

The 'HelloWorld.png' would be created in your project folder.

Modify Image

In this tutorial, you will learn how to modify an existing image and enlarge its size.

1. In the Program.cs file, include following namespaces -

```
using System;
using GrapeCity.Documents.Imaging;
```

2. Get file path for the image that needs to be enlarged.

```
var origImagePath = Path.Combine("Resources", "ImagesBis", "puffins-small.jpg");
```

3. Load original bitmap image

```
using (var origBmp = new GcBitmap(origImagePath))
{
```

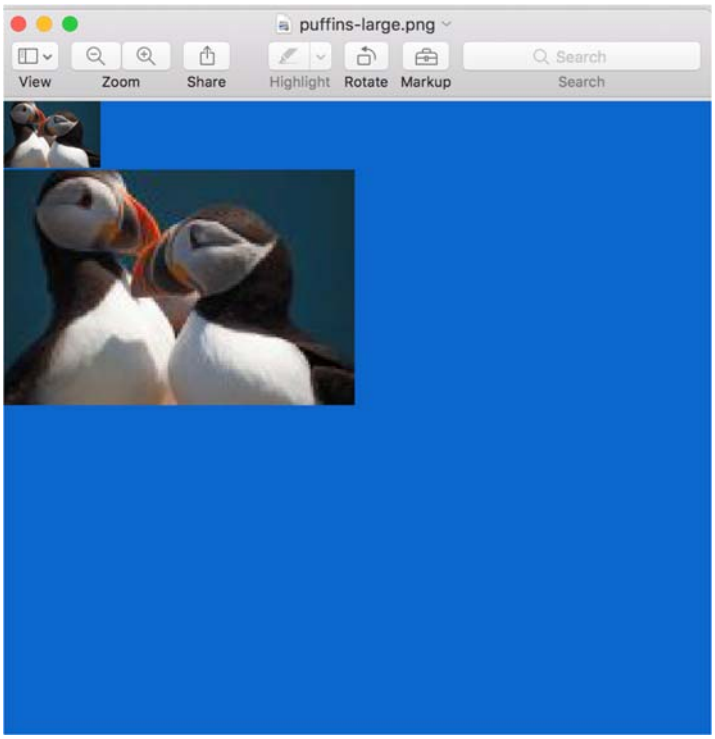
4. Enlarge the image

```
float k = 3.5f;
int twidth = (int)(origBmp.PixelWidth * k + 0.5f);
int theight = (int)(origBmp.PixelHeight * k + 0.5f);

using (var bmp = origBmp.Resize(twidth, theight,
                                InterpolationMode.Cubic))
```

5. Save the image

```
bmp.SaveAsPng("puffins-large.png");
}
```



The image shows the original image at top left, and the enlarged image.

Licensing

This is a BETA release of GcImaging. The product does not require a license to work without any limitations, but it will expire and stop working after Mar 31st, 2019. We are planning an official release of the product long before then. The beta expiry date would not apply if a valid license key is used after official release.

Demo Sample

Visit more GcImaging sample demos here.