Development guide

Apitron PDF Rasterizer for .NET

1. Introduction

Apitron PDF Rasterizer is a .NET component that performs quality conversion from PDF file to an image. It's 100% managed and doesn't require special manipulations to run with any .NET framework version starting from 2.0. Please read features section in order to get full description of its capabilities.

One purpose of this document is to give you an overview of the component, its features, limitations and applications.

Another one is to guide you through the Apitron PDF Rasterizer API and show possible ways of usage.

The document is divided by sections of interest and you may read either section you want first, no special reading order is assumed. Should you have any questions or probably want us to clarify something, please just send a note to support@apitron.com

2. Getting started

2.1 Development environment

Apitron PDF Rasterizer is compatible with any version of the .NET framework starting from .NET 2.0,
so all versions above that are fully supported(including Client Profile subsets).

2.2 Deployment & installation

Just unzip component package to a desired directory and add a reference to a component library,
Pick the version that fits your requirements from corresponding subfolder NET20, NET35, NET40 etc.
The only one assembly you need add reference to is Apitron.PDF.Rasterizer.Dll.

2.3 Licensing process

When component is not licensed it enforces an evaluation banner to be drawn on top of the rendered content of the resulting image. There are several license types which are listed and described in details on our website http://www.apitron.com/licensing.

In order to get your version licensed you have to go to our website, log in using personal account and request license activation. After that you'll be sent a license file created according to license kind you've bought .Please allow us some time to review the activation request, it will take max 24h to get the license file.

Having a license file the only thing you need is to apply it to a component, it's the simple step and you just have to choose whether you want to use our custom assembly attribute or set license manually using API.

2.3.1 Using custom assembly attribute

The component assembly contains **ApitronPDFRasterizerLicenseAttribute** class, and if you want to set license this way you have to apply it to the assembly that makes use of our component.
Typical way would look like:

1. Go to **AssemblyInfo.cs** file of your project (you may put attribute at any place, it's just for convenience)
2. Add the following code
   [assembly: ApitronPDFRasterizerLicenseAttribute(YOUR_LICENSE_STRING)]

   YOUR_LICENSE_STRING - is a text from a license file

2.3.2 Applying license manually

The component assembly contains **License** class, and if you want to set license manually you have to call its static method **SetLicense** and pass string from your license file.

3. Product description

3.1 PDF versions supported

PDF standard versions supported are: ALL versions. Files can be normal, linearized, password-protected, signed etc.

3.2. Features and limitations

We support rendering of the PDF content including:

- text(with embedded, externally linked, standard fonts)
- images, including masked ones
- annotation objects of various types
- complex paths and fills
- all blending modes
- tiling patterns
- shading patterns of type 1,2,3
- transparency groups(especially useful when you process files created with Adobe Illustrator)
- masked content
- various colorspaces

Things that are not listed above may work, however we do not guarantee that they will be processed correctly.

3.3. Output formats

We provide conversion to all image formats supported by .NET framework via System.Drawing.Bitmap class, therefore: BMP, JPG, PNG, TIFF etc.

3.4 Processing highlights

Key achievement of the current implementation is our rendering engine that is able to parallel some of the drawing operations and also supports advanced drawing operations defined by PDF standard.

File processing and parsing is also being done using all available processor cores so you application scales well when you add them.

We use continuous integration including automated testing environment that checks the component's solidness daily and helps us to quickly track and fix bugs. Lots of tests were created in order to give you the most tested and quality tool for your business and this test base is growing. We even developed internal testing tools so our QA department is well-armed.

4. Code samples

Complete code for samples listed below can be found in **Samples** folder inside the download package for the product. Code provided assumes you did all the necessary preparations(added references etc.) and just gives you an idea on how potentially the task could be done.

4.1. Convert PDF document to Bitmap

Below is a console program that converts specific PDF file to bitmap, a complete project can be found under:
[Download package]\Samples\ ConvertPdfToBitmap folder.

```
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;

using Apitron.PDF.Rasterizer;
using Apitron.PDF.Rasterizer.Configuration;

internal class Program
{
    private static void Main(string[] args)
    {
        // open and load the file
        using (FileStream fs = new FileStream(@"Documents\testfile.pdf", FileMode.Open))
        {
            // this objects represents a PDF document
            Document document = new Document(fs);

            // process and save pages one by one
            for (int i = 0; i < document.Pages.Count; i++)
            {
                Page currentPage = document.Pages[i];

                // we use original page's width and height for image as well as default
                           rendering settings
                 using (Bitmap bitmap = currentPage.Render((int)currentPage.Width,
                                  (int)currentPage.Height, new RenderingSettings()))
                 {
                     bitmap.Save(string.Format("{0}.png", i), ImageFormat.Png);
                 }
             }
         }
     }
 }
```

4.2. Convert PDF to TIFF

Below is a console program that converts specific PDF file to tiff, a complete project can be found under:
[Download package]\Samples\ ConvertPdfToTiff folder.

```
using System.IO;
using Apitron.PDF.Rasterizer;
using Apitron.PDF.Rasterizer.Configuration;

internal class Program
{
    private static void Main(string[] args)
    {
        // open and load the file
        using (FileStream fs = new FileStream(@"Documents\testfile.pdf", FileMode.Open),
                    fsOut = File.Create("out.tiff"))
        {
                // this objects represents a PDF document
             Document document = new Document(fs);

             // save to tiff using CCIT4 compression, black and white tiff.
             // set the DPI to 144.0 for this sample, so twice more than default PDF dpi
setting.
             TiffRenderingSettings settings = new
TiffRenderingSettings(TiffCompressionMethod.LZW, 144, 144);

             document.SaveToTiff(fsOut, settings);
         }
    }
}
```

4.3. Viewing information about PDF document

Below is a console program that reads information about specific PDF file, like author, title etc.A complete project can be found under:
[Download package]\Samples\ ViewDocumentInformation folder.

```
using System.IO;
using Apitron.PDF.Rasterizer;

internal class Program
{
    private static void Main(string[] args)
    {
        // open and load the file
        using (FileStream fs = new FileStream(@"Documents\testfile.pdf", FileMode.Open))
        {
            // this objects represents a PDF document
            Document document = new Document(fs);

            // get the object that holds information about document
            DocumentInfo info = document.DocumentInfo;

            // print out desired fields
            Console.WriteLine(string.Format("Document title:{0}", info.Title));
            Console.WriteLine(string.Format("Page count:{0}", document.Pages.Count));
            Console.WriteLine(string.Format("Document author:{0}", info.Author));
            Console.WriteLine(string.Format("Created with:{0}", info.Producer));

            // dates by default are returned as they are written in PDF file,
            // to give you the full control over stored information.
            // helper method used below, converts PDF date representation to DateTime
object.
            // you may use it, if you don't need special processing
            Console.WriteLine( string.Format("Creation date:{0}",
DocumentInfo.ParsePdfDate( info.CreationDate ) ) );

            // read further if needed..

            Console.ReadLine();
        }
    }
}
```

4.4. Extracting font information from PDF files

Sometimes you need to know how to find out whether font is embedded or referenced, below is a console program that enumerates fonts used in document reports this information. Sample is located under
[Download package]\Samples\ ExtractFontInformation folder.

```
using System.IO;
using Apitron.PDF.Rasterizer;
using Apitron.PDF.Rasterizer.Fonts;

internal class Program
{
    private static void Main(string[] args)
    {
        // open and load the file
        using (FileStream fs = new FileStream(@"Documents\testfile.pdf", FileMode.Open))
        {
            // this objects represents a PDF document
            Document document = new Document(fs);

            // enumerate fonts used in document
            foreach (Font font in document.Fonts)
            {
                // print out the font name, its type and state
                Console.WriteLine(string.Format("Font name: {0}", font.Name));
                Console.WriteLine( string.Format("Font type: {0}",
                     Enum.GetName( typeof( FontType ), font.Type)));
                Console.WriteLine(string.Format("Font state: {0}",
                            Enum.GetName(typeof(FontState), font.State)));
            }

            Console.ReadLine();
        }
    }
}
```

4.5 Advanced conversion using rendering settings

RenderingSettings class allows to tune up rendering engine so it produces the result that fits you needs.You may set desired background color, page rotation angle, scaling mode and other settings. Sample below demonstrates how to use this class. It can be found under [Download package]\Samples\ RenderingSettingsUsage folder.

```
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;
using Apitron.PDF.Rasterizer;
using Apitron.PDF.Rasterizer.Configuration;

internal class Program
{
    private static void Main(string[] args)
    {
        // open and load the file
        using (FileStream fs = new FileStream(@"Documents\testfile.pdf", FileMode.Open))
        {
            // this objects represents a PDF document
            Document document = new Document(fs);

            // initialize settings object
            RenderingSettings settings = new RenderingSettings();

            // i want a special background for my pages
            settings.BackgroundColor = Color.LawnGreen.ToArgb();

            // annotations objects like notes, will be drawn
            settings.DrawAnotations = true;

            // images will be drawn as well
            settings.DrawImages = true;

            // text on
            settings.DrawText = true;

            // i want page to be turned 90 degrees clockwise
            settings.RotationAngle = RotationAngle.Rotate90;

            // i want page content to fit undistorted, so let's preserve an aspect ratio
            settings.ScaleMode = ScaleMode.PreserveAspectRatio;

            // process and save pages one by one
            for (int i = 0; i < document.Pages.Count; i++)
            {
                Page currentPage = document.Pages[i];

                // we use original page's width and height for image as well as default
                       rendering settings
                using (Bitmap bitmap = currentPage.Render((int)currentPage.Width,
                             (int)currentPage.Height, settings))
                {
                    bitmap.Save(string.Format("{0}.png", i), ImageFormat.Png);
                }
            }
        }
    }
}
```