Development guide

Apitron PDF Kit for .NET

1. Introduction

Apitron PDF Kit is a .NET component that allows you to do whatever you want with PDF files. Add text, images, sign documents and many more. You can also edit existing content using its friendly and easy to use API. It's 100% managed code 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 Kit 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 Kit is compatible with any version of the .NET framework starting from .NET 2.0, so all versions above that are fully supported.

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 NET v2.0, NET v3.5, NET v4.0 etc. The only assembly you need to add reference to is Apitron.PDF.Kit.dll.

2.3 Licensing process

When component is not licensed it enforces an evaluation banner to be added on top of the content for all pages. There are several license types which are listed and described in details on our website, see http://www.apitron.com/licensing for more information.

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 a very 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 **ApitronPDFKitLicenseAttribute** 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: ApitronPDFKitLicenseAttribute (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 Architecture and programming model

The Apitron PDF Kit consists of two main parts implementing different approaches to PDF content creation and manipulation. These parts are described below:

- **FixedDocument** – it's the core class for loading existing PDF documents or creating new from scratch. Its implementation, including other referenced classes, has the structure similar to the one defined in PDF specification. It follows the specification as close as possible making you able to use all features which are available to PDF creators.
- **FlowDocument** - makes you able to define compact, stylish document templates and produce stunning PDF documents in minutes. It has own content and styling model similar to HTML + CSS and is very easy to use. It has a list of markup elements, e.g. textblock, image, grid, etc. which can be all styled using style objects. Bookmarks, links, and interactive form controls are supported as well. Furthermore, such templates called "flowdocuments" can be exported to XML, thus separating the markup from content that can be later added by loading the template from XML and putting content into placeholders.

Having these parts we designed them in such a way that they can easily interoperate with each other, so it's possible to put markup elements into *FixedDocument* and use raw pdf content in *FlowDocument*.

3.3. Features and limitations

We support many possible PDF content manipulations scenarios, below are a few things which worth mentioning. Using Apitron PDF Kit for .NET you may:

- Extract, modify and add graphics (text, images, drawings)
- Create soft masks, image masks, stencil masks, color masks.
- Split or merge PDF documents
- Fill or create PDF forms
- Add or remove document fields
- Examine resources within a document - fonts, embedded files
- Digitally sign and check existing signatures on PDF documents
- Protect document with passwords
- Work with navigation objects, e.g. create bookmarks or links
- Fully control document's annotations
- Fully control various PDF actions
- Use all fonts defined by specification
- Use various colorspaces and color profiles, e.g. you may draw in CMYK, or gray, or whatever colorspace you like.
- Use flow layout model for document creation, see FlowDocument class
- Save or load FlowDocument model to/from XML, suitable for creation of document templates

3.4 Processing highlights

File processing and parsing is being done using all available processor cores so you application scales well when you add them. Our state of art parsing engine is a true gem we that were developing for a long time. We are passionate about PDF and have enough industry experience to make our component the best on a market.

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.

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 things needed for compilation(added references etc.) and just gives you an idea on how potentially the task could be done.

4.1. Add simple text sample (fixed layout)

Adding pages with trivial text to a document is extremely simple. Below is a console program that creates PDF file with text, a complete project can be found under:
[Download package]\Samples\FixedLayoutSamples\Text\Simple Text folder.

```
// create new PDF file
using (FileStream fs = new FileStream(@"\OutputDocuments\SimpleText.pdf", FileMode.Create))
{
	// this object represents a PDF fixed document
	FixedDocument document = new FixedDocument();
	Page page = new Page(new PageBoundary(Boundaries.A4));
	document.Pages.Add(page);

	// create text object based on standard fonts
	TextObject text = new TextObject(StandardFonts.Helvetica, 16);
	text.SetTextMatrix(1, 0, 0, 1, 80, 700);
	text.AppendNewLineText("celebrate");
	page.Content.AppendText(text);
	text.SetTextMatrix(1, 0, 0, 1, 80, 670);
	text.SetFont(StandardFonts.TimesBoldItalic, 16);
	text.AppendNewLineText("fanciful");
	page.Content.AppendText(text);
	text.SetTextMatrix(1, 0, 0, 1, 80, 640);
	text.SetFont(StandardFonts.CourierBoldOblique, 16);
	text.AppendNewLineText("groovy");

	page.Content.AppendText(text);
	document.Save(fs);
}

Process.Start(@"\OutputDocuments\SimpleText.pdf");
```

4.2. Add simple image sample (fixed layout)

Adding pages with image XObject to a document is very simple. Below is a console program that creates PDF file with registered image resource, a complete project can be found under: *[Download package]\Samples\FixedLayoutSamples\ Images\Add and draw images* folder.

```
using (FileStream fs = new FileStream( @"\OutputDocuments\AddAndDrawImages.pdf",
FileMode.Create, FileAccess.Write ))
{
	// this object represents a PDF fixed document
	FixedDocument document = new FixedDocument();

	string resourceID = "IMG0";
	FixedLayout.Resources.XObjects.Image image = new
	FixedLayout.Resources.XObjects.Image(resourceID, @"\OutputDocuments\image.jpg");
	document.ResourceManager.RegisterResource( image );
	Boundary boundary = new Boundary(0, 0, image.Width + 20, image.Height + 60);
	Page page = new Page(new PageBoundary(boundary));
	page.Content.AppendImage(resourceID, 10, 50, image.Width, image.Height);
	document.Pages.Add(page);
	document.Save(fs);
}
```

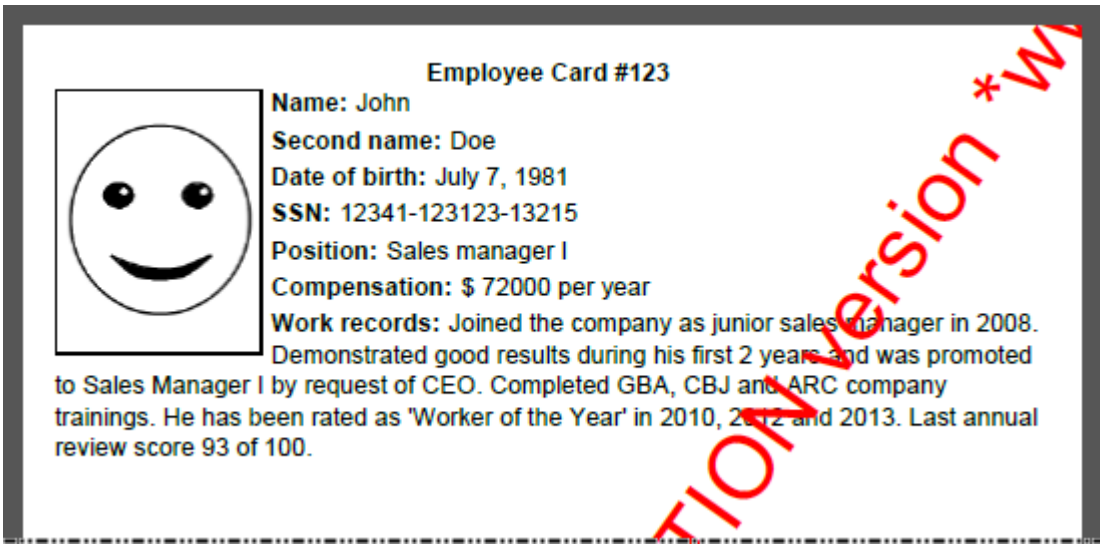
4.3 Create Employee Card sample (flow layout)

Creation of PDF documents using FlowDocument is very easy and straightforward, a complete project can be found under: *[Download package]\Samples\FlowLayoutSamples\ EmployeeCard* folder.

```
/// This sample demonstrates how to create a simple employee card document.
/// create resource manager and register image resource
ResourceManager resourceManager = new ResourceManager();
resourceManager.RegisterResource(new
Apitron.PDF.Kit.FixedLayout.Resources.XObjects.Image("photo",
System.IO.Path.Combine("data","johndoe.png")));

// create flow document
FlowDocument document = new FlowDocument();
document.Margin = new Thickness(Length.FromInches(0.25));

// create style for personal photo
document.StyleManager.RegisterStyle("image#employeePhoto", new Style{Float=Float.Left, Border=new
Border(2), BorderColor=RgbColors.Black, Margin= new Thickness(0,0,5,5)});
// create style for header
document.StyleManager.RegisterStyle("textblock.header", new Style{Display=Display.InlineBlock,
Align=Align.Center, Margin=new Thickness(0,0,0,5)});
// create style for data labels
document.StyleManager.RegisterStyle("textblock.label",new Style{Font = new
Font(StandardFonts.HelveticaBold, 14), Margin=new Thickness(0, 0, 5, 0)});
// create style for text data
document.StyleManager.RegisterStyle("textblock.data",new Style{ Font = new
Font(StandardFonts.Helvetica, 14) });
// create style for br element that goes after textblock
document.StyleManager.RegisterStyle("textblock+br", new Style{Height = 3});

// fill the document
document.Add(new TextBlock("Employee Card #123"){Class = "header label"});

document.Add(new Image("photo"){Id = "employeePhoto"});

document.Add(new TextBlock("Name:"){Class = "label"});
document.Add(new TextBlock("John") {Class = "data" });
document.Add(new Br());

document.Add(new TextBlock("Second name:"){Class = "label"});
document.Add(new TextBlock("Doe") {Class = "data"});
document.Add(new Br());

document.Add(new TextBlock("Date of birth:"){Class = "label"});
document.Add(new TextBlock("July 7, 1981") {Class = "data"});
document.Add(new Br());

document.Add(new TextBlock("SSN:"){Class = "label"});
document.Add(new TextBlock("12341-123123-13215") {Class = "data"});
document.Add(new Br());

document.Add(new TextBlock("Position:"){Class = "label"});
document.Add(new TextBlock("Sales manager I") {Class = "data"});
document.Add(new Br());

document.Add(new TextBlock("Compensation:") {Class = "label"});
document.Add(new TextBlock("$ 72000 per year") {Class = "data"});
document.Add(new Br());

document.Add(new TextBlock("Work records:"){Class = "label"});
```

```
document.Add(new TextBlock("Joined the company as junior sales manager in 2008." +
" Demonstrated good results during his first 2 years and was promoted to Sales Manager I by
request of CEO." +
" Completed GBA, CBJ and ARC company trainings. He has been rated as 'Worker of the Year' in
2010, 2012 and 2013. Last annual review score 93 of 100. ") { Class = "data" });
document.Add(new Br());

// save and open
using (Stream stream = File.Create(@"\OutputDocuments\EmployeeCard.pdf";))
{
  document.Write(stream, resourceManager, new PageBoundary(Boundaries.A4));
}
```

The result of execution is below:



Pic.1 Generated Employee Card Document