How to resize PDF pages and stamp PDF documents

Written by Apitron Documentation Team

Introduction

Many custom workflows which are in use in small and big companies implement incoming PDF documents processing and stamping for tracking, archiving, and later use. These docs can be of any origin: received by emails, scanned by the personnel from hard copies etc.

Manual processing takes significant amount of time, and in this article we'll show how to speed up this process by creating an app for automatic PDF page resizing and stamping.

The app

Consider the following simple program:

```
namespace ResizeAndStampPDF
{
    class Program
    {
        static void Main(string[] args)
        {
            if (args != null && args.Length > 0)
            {
                string destinationPath =
                   System.IO.Path.GetFileNameWithoutExtension(args[0]) + "_stamped.pdf";

                PageStamper.Stamp(args[0],destinationPath,"FF6173","949124","Approved");

                Process.Start(destinationPath);
            }
        }
    }
}
```

It uses the *PageStamper.Stamp()* method to perform the stamping task by passing doc's attributes. This class uses Apitron PDF Kit .NET component and also demonstrates its Fixed and Flow layout APIs in action (see this blog post to read about the difference between them). See the next section for the details.

PDF Stamping implementation

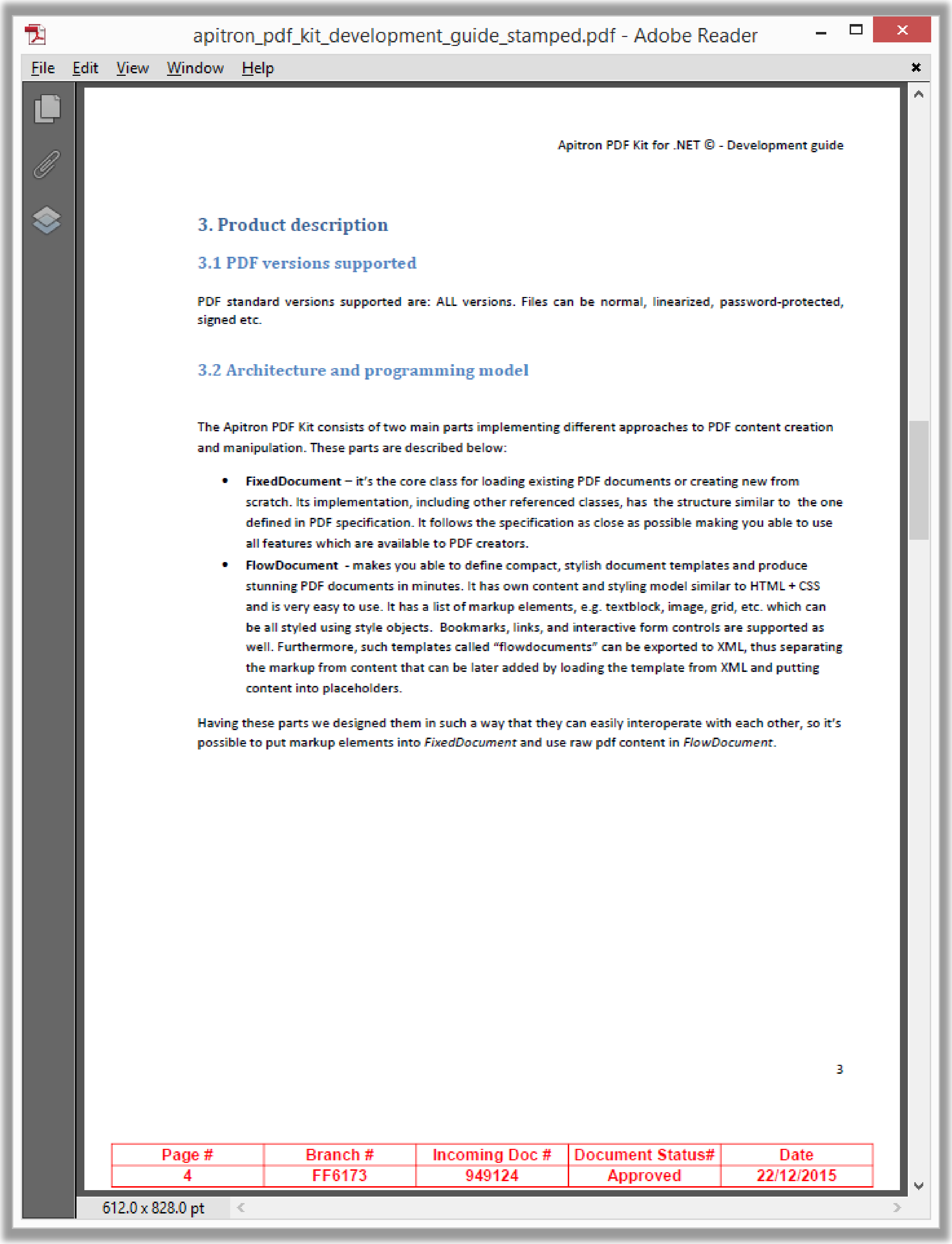
Class *PageStamper* listed below transforms the page by expanding it down and adding a stamp onto this area.

```
/// <summary>
/// Resizes PDF page and adds custom stamp at the bottom.
/// </summary>
static class PageStamper
{
    private const int stampHeight = 36;
    private  const int sectionMargin = 40;

    public static void Stamp(string sourcePath, string destinationPath, string branchNo,
        string incomingNo, string documentStatus)
    {
        // open file
        using (Stream stream = File.Open(sourcePath, FileMode.Open, FileAccess.ReadWrite))
        {
            // load pdf document from stream
            using (FixedDocument originalDoc = new FixedDocument(stream))
            {
                int pageNumber = 1;
                // process pages
                foreach (Page page in originalDoc.Pages)
                {
                    // append the required stamp
                    Section stampSection = CreateStampSection(page, branchNo, incomingNo,
                       documentStatus, pageNumber++);
                    // resize the page
                    ResizePage(page);
                    // save state to avoid any problems with possible graphics state change
                    page.Content.SaveGraphicsState();
                    // apply transformations to properly position the stamp
                    ApplyTransformations(page, stampSection);
                    // add section object to the page
                    page.Content.AppendContentElement(stampSection,
                        stampSection.Width.Value, stampSection.Height.Value);
                    // restore the original state
                    page.Content.RestoreGraphicsState();
                }
                // save changes to PDF
                using (Stream outputStream = File.Create(destinationPath))
                {
                    originalDoc.Save(outputStream);
                }
            }
        }
    }

    /// <summary>
    /// Resizes PDF page, taking into account its rotation property.
    /// </summary>
    /// <param name="page"></param>
    private static void ResizePage(Page page)
    {
        Boundary mediaBox = ResizeBoundary(page.Boundary.MediaBox, page.Rotate);
        page.Resize(new PageBoundary(mediaBox));
    }
```

```csharp
private static Section CreateStampSection(Page page, string branchNo, string incomingNo,
    string documentStatus, int pageNumber)
{
    // create section content element and set its size
    Section stampSection = new Section();
    stampSection.Height = stampHeight;

    if(page.Rotate == PageRotate.Rotate0 || page.Rotate == PageRotate.Rotate180)
    {
        stampSection.Width = page.Boundary.MediaBox.Width - sectionMargin;
    }
    else
    {
        stampSection.Width = page.Boundary.MediaBox.Height - sectionMargin;
    }

    // create grid element
    Grid grid = new Grid(Length.Auto, Length.Auto, Length.Auto, Length.Auto, Length.Auto);
    grid.Font = new Font(StandardFonts.HelveticaBold, 12);
    grid.Color = RgbColors.Red;
    grid.InnerBorderColor = RgbColors.Red;
    grid.InnerBorder = new Border(1);
    grid.Width = Length.FromPercentage(100);

    // add header row
    grid.Add(new GridRow( new TextBlock("Page #"),
        new TextBlock("Branch #"),
        new TextBlock("Incoming Doc #"),
        new TextBlock("Document Status#"),
        new TextBlock("Date"))
        { Align = Align.Center });

    // add data row
    grid.Add(new GridRow( new TextBlock(pageNumber.ToString()),
        new TextBlock(branchNo),
        new TextBlock(incomingNo),
        new TextBlock(documentStatus),
        new TextBlock(DateTime.Now.ToString("dd/MM/yyyy")))
        { Align = Align.Center });

    stampSection.Add(grid);
    return stampSection;
}

/// <summary>
/// Applies transformations to generated content considering initial page rotation.
/// </summary>
/// <param name="page">The page to transform.</param>
/// <param name="section">Section used </param>
private static void ApplyTransformations(Page page, Section section)
{
    switch (page.Rotate)
    {
        case PageRotate.Rotate90:
        {
            page.Content.SetRotate(Math.PI/2.0f);
            // set current position on page
            page.Content.Translate((page.Boundary.MediaBox.Height - section.Width.Value)/2.0,
                -page.Boundary.MediaBox.Right);
            break;
        }


      // code continues on next page
```

```csharp
                case PageRotate.Rotate180:
                {
                    page.Content.SetRotate(Math.PI);
                    // set current position on page
                    page.Content.Translate(
                        -page.Boundary.MediaBox.Width +
                        ((page.Boundary.MediaBox.Width - section.Width.Value)/2.0),
                        -page.Boundary.MediaBox.Height);
                    break;
                }
                case PageRotate.Rotate270:
                {
                    page.Content.SetRotate(-Math.PI/2.0f);
                    // set current position on page
                    page.Content.Translate(
                        -page.Boundary.MediaBox.Height +
                        (page.Boundary.MediaBox.Height - section.Width.Value)/2.0,
                        page.Boundary.MediaBox.Left);

                    break;
                }
                case PageRotate.Rotate0:
                default:
                {
                    // set current position on page
                    page.Content.Translate((page.Boundary.MediaBox.Width - section.Width.Value)/2.0,
                        page.Boundary.MediaBox.Bottom);
                    break;
                }
            }
        }

        /// <summary>
        /// Resizes passed boundary if it's not null, otherwise returns null.
        /// </summary>
        /// <param name="boundary">Boundary to resize.</param>
        /// <param name="rotation">Page rotation.</param>
        /// <param name="heightDelta">The height delta, defaut is 36.</param>
        /// <returns>New boundary the <paramref name="boundary"/> is not null, otherwise null.</returns>
        private static Boundary ResizeBoundary(Boundary boundary, PageRotate rotation,
            double heightDelta = stampHeight)
        {
            if (boundary == null)
            {
                return null;
            }
            switch (rotation)
            {
                case PageRotate.Rotate90:
                    return new Boundary(boundary.Left, boundary.Bottom, boundary.Right + heightDelta,
                        boundary.Top);
                case PageRotate.Rotate180:
                    return new Boundary(boundary.Left, boundary.Bottom, boundary.Right,
                        boundary.Top + heightDelta);
                case PageRotate.Rotate270:
                    return new Boundary(boundary.Left - heightDelta, boundary.Bottom,
                        boundary.Right + heightDelta, boundary.Top);
                case PageRotate.Rotate0:
                default:
                    return new Boundary(boundary.Left, boundary.Bottom - heightDelta, boundary.Right,
                        boundary.Top);
            }
        }
    }
```

The resulting document, transformed by the code above, looks as follows:


apitron_pdf_kit_development_guide_stamped.pdf - Adobe Reader

Apitron PDF Kit for .NET © - Development guide

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

Page #	Branch #	Incoming Doc #	Document Status#	Date
4	FF6173	949124	Approved	22/12/2015

612.0 x 828.0 pt

Pic. 1 Transformed PDF document with stamp

Conclusion

This sample shows how to combine fixed and flow layout API together and get the best form both. We open the document using Fixed layout API and add a Section content element from Flow layout API after resizing. It helps us to quickly generate the stamp grid, and avoid many manual drawing operations.

The actual resizing magic happens in *Page.Resize*() method which repositions the content according to its initial state.

This project is a small demo of what Apitron PDF Kit for .NET can do with PDF and if you'd like to know more, then our free book is at your service. The complete code for this example can be found in our GitHub repo.