How to programmatically delete, edit or replace content in PDF documents

Written by Apitron Documentation Team

Introduction

Replacing, editing or deleting content from PDF documents programmatically is not a trivial task and requires expert knowledge of the format and internal structures to be implemented from scratch. Luckily, we made it much easier for you by introducing native support for these operations. You can examine document's content page by page and change the things you need without any significant efforts. In this article we'll demonstrate how to implement text and image replacement or editing, removing contents from the desired area or region, resources replacement, graphics paths alteration, and getting content elements' boundaries.

Replacing text and images

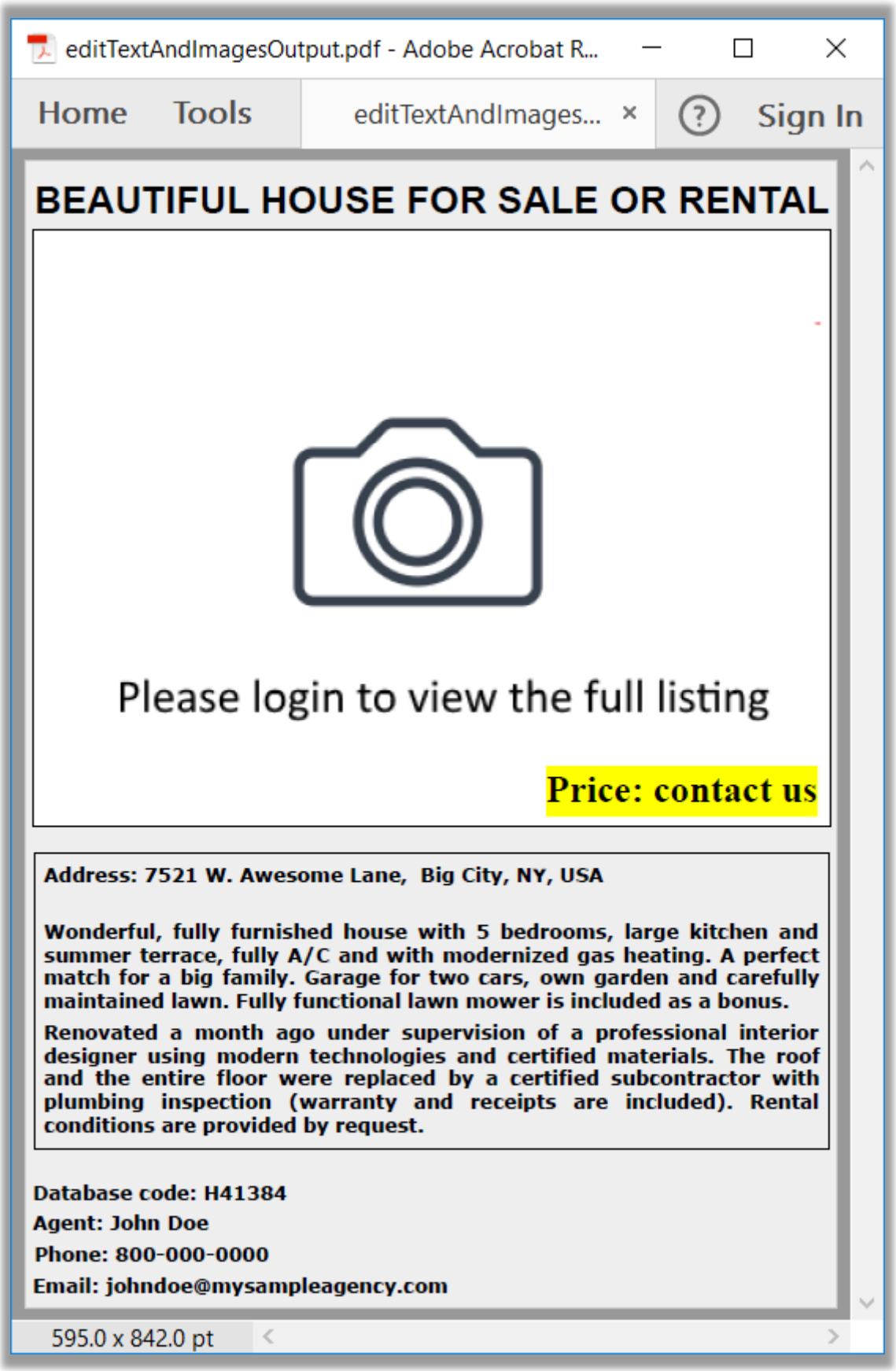
Let's assume you're developing a web-based solution for a real estate agency and you need to process advertisements stored as PDF documents. One of them could look as below:

Pic. 1 Sample advertisement stored as PDF

But the complete listing should only be accessible to the logged in customers, while you still want the ad to be viewable by other users but with some restrictions that include price and the photo of the object. One of solutions is to generate it dynamically. Here is the code:

```
static void Main(string[] args)
{
       ReplaceTextAndImages("../../../data/advertisement.pdf", "$","Price: contact us",
              "../../../data/replacement.png");
}

private static void ReplaceTextAndImages(string inputFilePath, string oldText,
       string newText, string replacementImagePath)
{
    using (Stream inputStream = File.Open(inputFilePath, FileMode.Open, FileAccess.Read))
    {
        using (FixedDocument doc = new FixedDocument(inputStream))
        {
            // add the replacement image to document's resources
            doc.ResourceManager.RegisterResource(new Image("replacement_image",
               replacementImagePath, true));

            // enumerate content elements found on document's first page
            foreach (IContentElement element in doc.Pages[0].Elements)
            {
                // handle the text element case
                if (element.ElementType == ElementType.Text)
                {
                    TextContentElement textElement = element as TextContentElement;
                    if (textElement != null)
                    {
                        // go thought all the text segments and replace
                        // the segment that contains the sample text
                        foreach (TextSegment textSegment in textElement.Segments)
                        {
                            if (textSegment.Text.Contains(oldText))
                            {
                                TextObject newTextObject =
                                  new TextObject(textSegment.FontName,textSegment.FontSize);
                                newTextObject.AppendText(newText);
                                textSegment.ReplaceText(0, textSegment.Text.Length, newTextObject);
                            }
                        }
                    }
                } // handle image case
                else if (element.ElementType == ElementType.Image)
                {
                    ImageContentElement imageElement = element as ImageContentElement;

                    if (imageElement != null)
                    {
                        // just replace the image with new one using
                        // registered resource, removing old one
                        imageElement.Replace("replacement_image", true);
                    }
                }
            }

            // save modified file
            using (Stream outputStream = File.Create(outputFileName))
            {
                doc.Save(outputStream);
            }
        }
    }

    Process.Start(outputFileName);
}
```

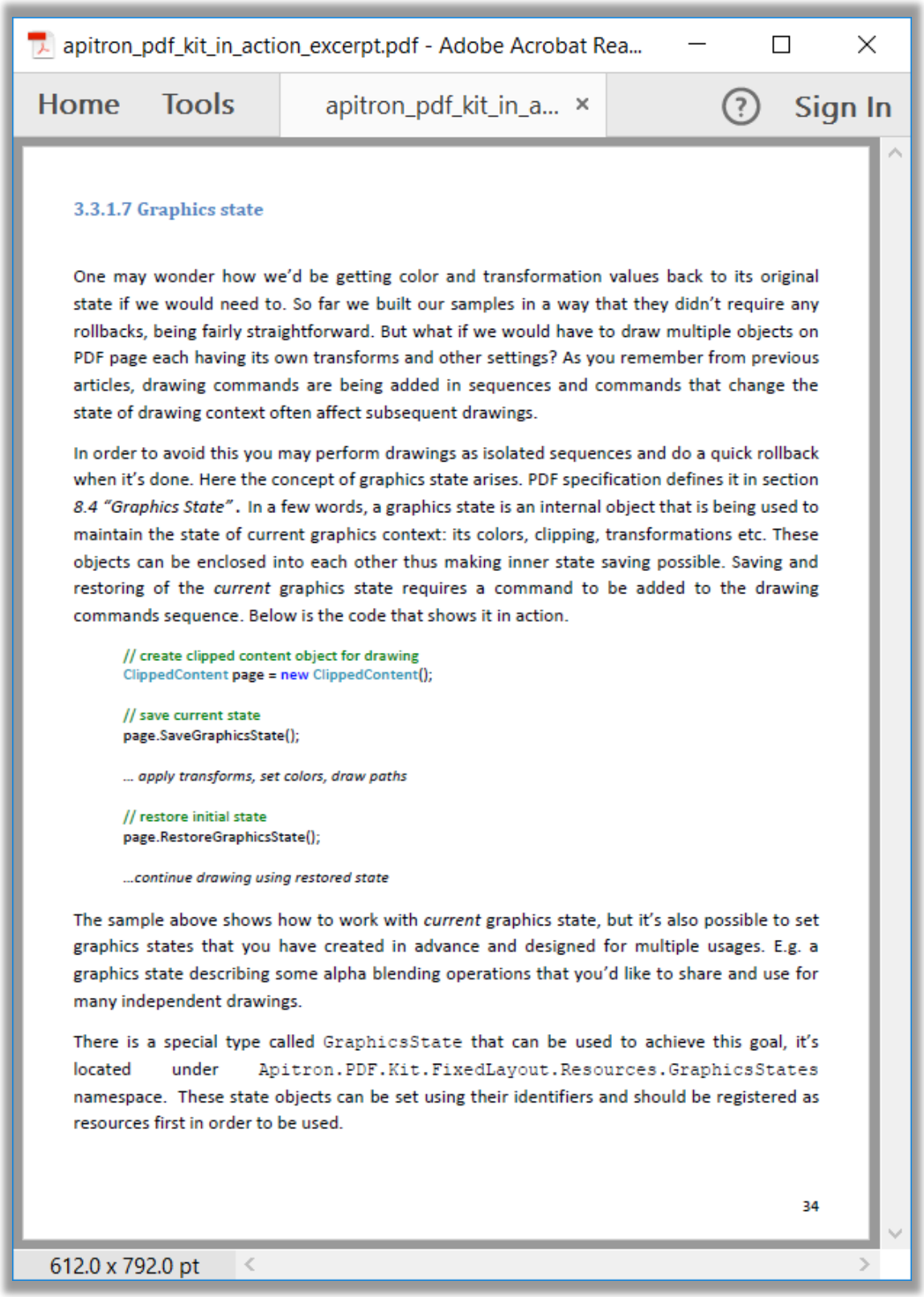
And the resulting file produced by this code is shown below:



Pic. 2 Edited PDF document

Content deletion

Let's say you have a document shown below and would like to remove all content that intersects with an arbitrary rectangular region.

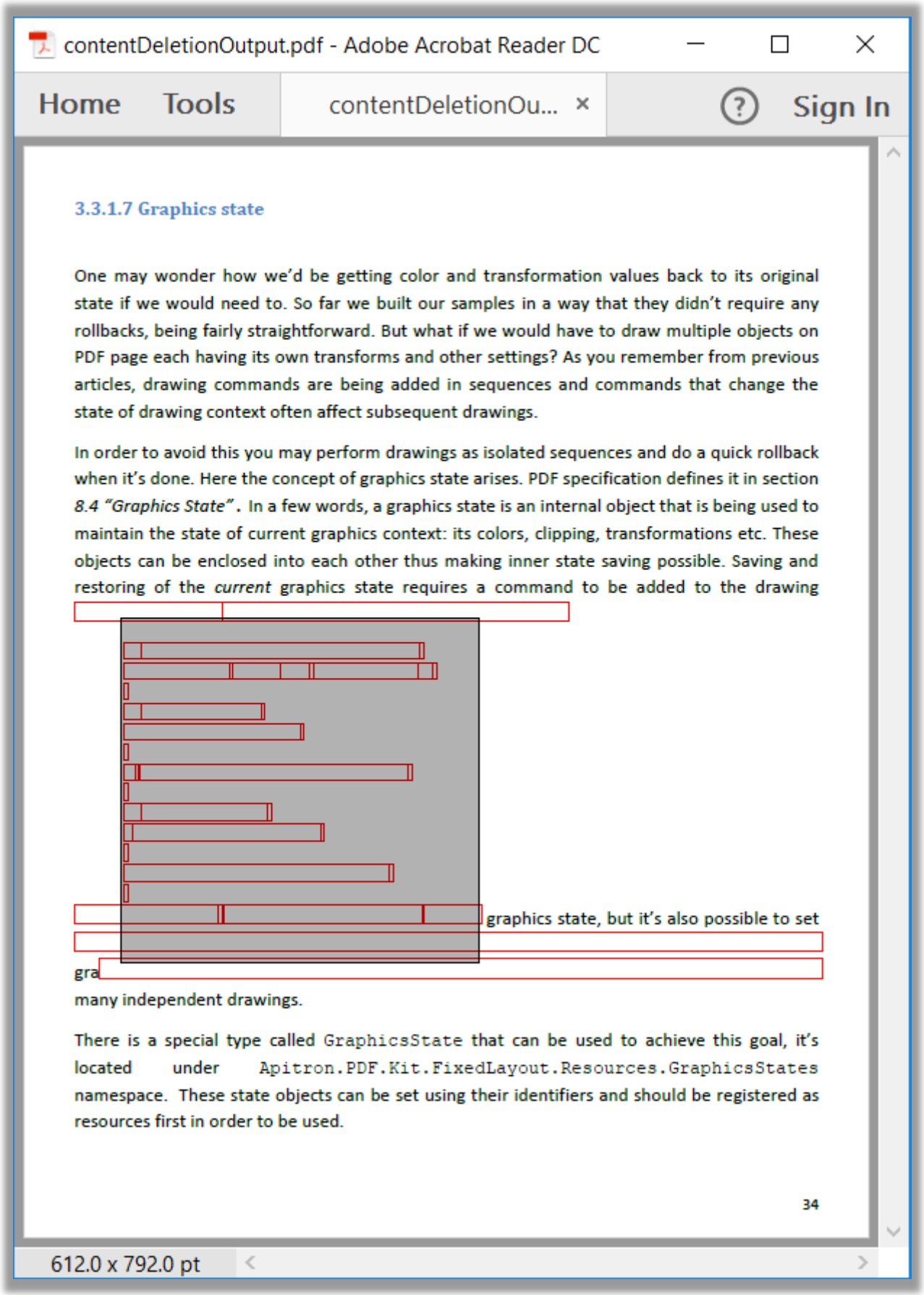


Pic. 3 Sample document for content removal

Here is the code that does the job, it also highlights the elements that were removed using their calculated boundaries:

```
static void Main(string[] args)
{
    RemoveContentInRect("../../../data/apitron_pdf_kit_in_action_excerpt.pdf",
       new Boundary(70, 200, 330, 450));
}

private static void RemoveContentInRect(string inputFilePath, Boundary redactionRect)
{
    using (Stream inputStream = File.Open(inputFilePath, FileMode.Open, FileAccess.Read))
    {
        using (FixedDocument doc = new FixedDocument(inputStream))
        {
            doc.ResourceManager.RegisterResource(
              new GraphicsState("myGraphicsState") {CurrentNonStrokingAlpha = 0.3});

            // enumerate content elements found on document's first page
            Page firstPage = doc.Pages[0];

            firstPage.Content.SaveGraphicsState();
            firstPage.Content.SetDeviceStrokingColor(new []{1.0,0,0});

            foreach (IContentElement element in firstPage.Elements)
            {
                // remove elements falling into the deletion region
                // even if they just overlap
                if (element.ElementType == ElementType.Text)
                {
                    TextContentElement textElement = (TextContentElement) element;

                    foreach (TextSegment segment in textElement.Segments)
                    {
                        if (RectsOverlap(redactionRect, segment.Boundary))
                        {
                            firstPage.Content.StrokePath(Path.CreateRect(segment.Boundary));
                            segment.Remove();
                        }
                    }
                }
                else if (!RectsOverlap(redactionRect, element.Boundary))
                {
                    firstPage.Content.StrokePath(Path.CreateRect(element.Boundary));
                    element.Remove();
                }
            }

            // highlight deletetion region
            firstPage.Content.SetGraphicsState("myGraphicsState");
            firstPage.Content.SetDeviceStrokingColor(new []{0.0});
            firstPage.Content.SetDeviceNonStrokingColor(new []{0.0});
            firstPage.Content.FillAndStrokePath(Path.CreateRect(redactionRect));
            firstPage.Content.RestoreGraphicsState();

            // save modified file
            using (Stream outputStream = File.Create(outputFileName))
            {
                doc.Save(outputStream);
            }
        }
    }
}
public static bool RectsOverlap(Boundary a, Boundary b)
{
    return (a.Left < b.Right && a.Right> b.Left && a.Bottom<b.Top && a.Top>b.Bottom);
}
```

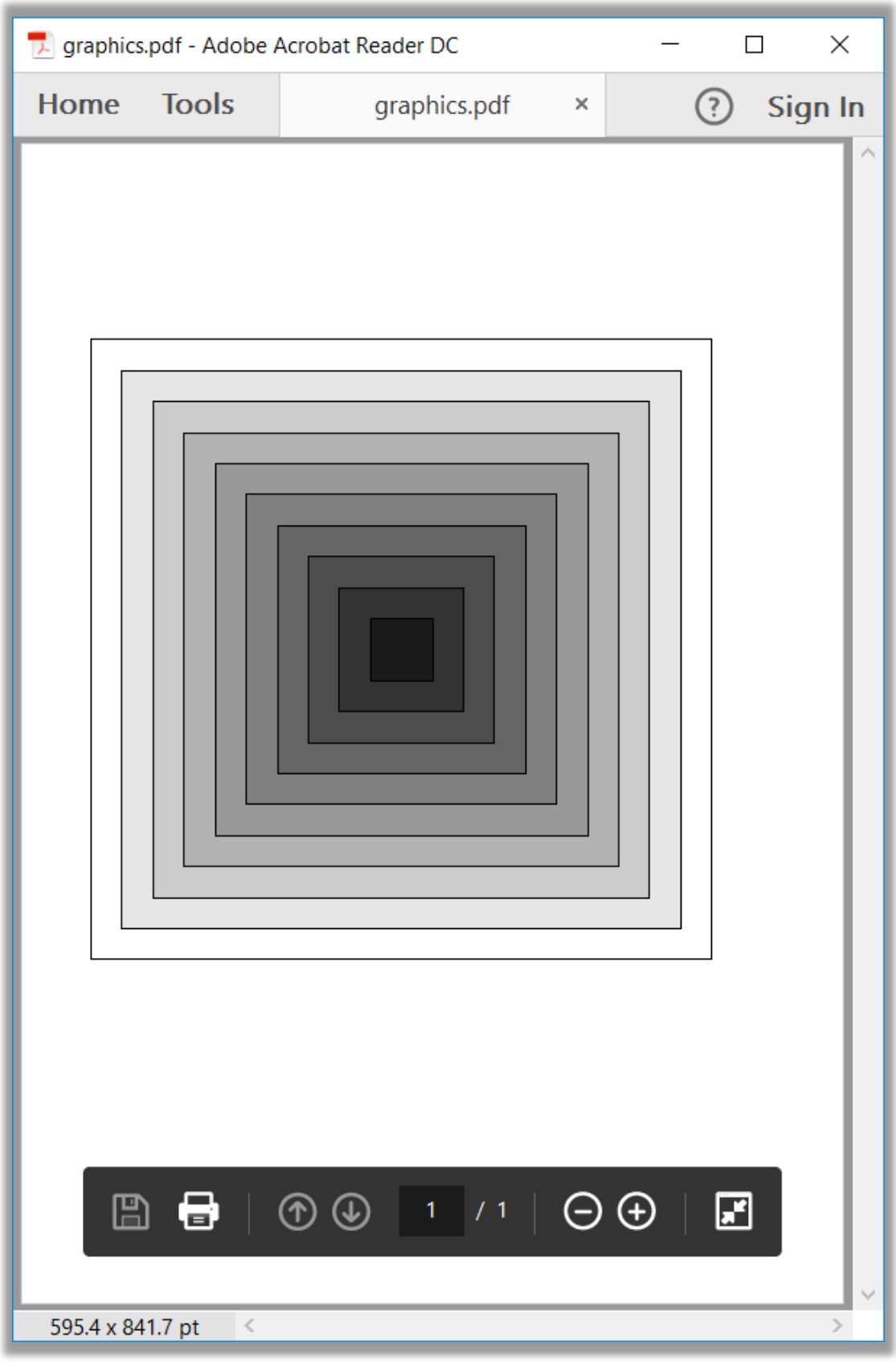
Resulting document is demonstrated below:



Pic. 4 Document with partially removed content

Changing existing drawings or graphics paths

If you have a drawing you would like to alter there is an API for that as well. You can also prepend or append PDF content to it, scale, translate or delete. Here is our sample file:



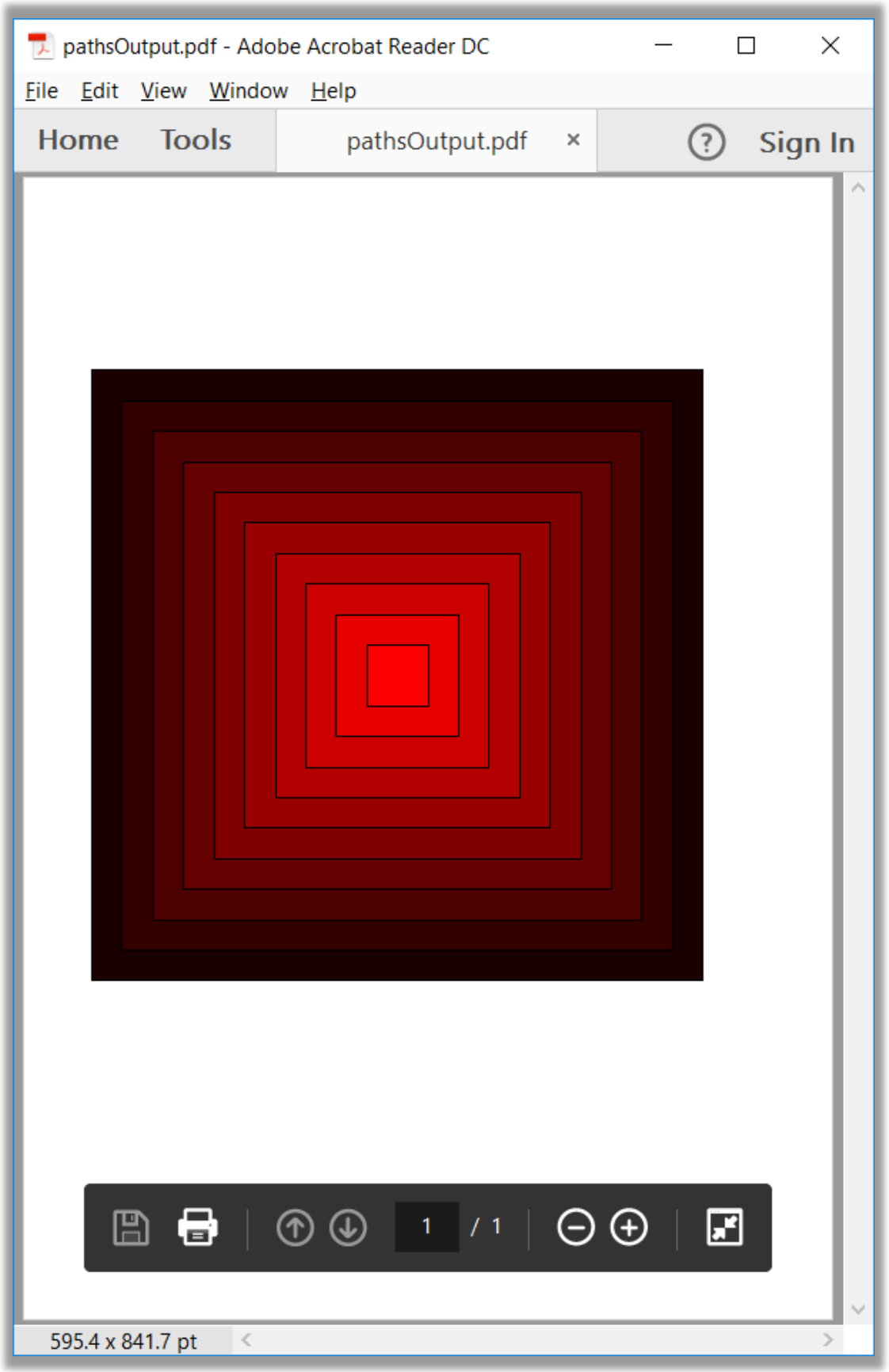
Pic. 5 PDF document with vector drawing

And our code that changes it a bit by altering non stroking colors for all found paths:

```
static void Main(string[] args)
{
    ReplacePaths("../../../data/graphics.pdf");
}

private static void ReplacePaths(string inputFilePath)
{
    using (Stream inputStream = File.Open(inputFilePath, FileMode.Open, FileAccess.Read))
    {
        using (FixedDocument doc = new FixedDocument(inputStream))
        {
            double colorComponent = 0;
            double colorDelta = 0.1;

            // enumerate content elements found on document's first page
            foreach (IContentElement element in doc.Pages[0].Elements)
            {
                // change the fill color of each found drawing
                if (element.ElementType == ElementType.Drawing)
                {
                    DrawingContentElement drawingElement = (DrawingContentElement) element;
                    drawingElement.SetNonStrokingColor(new double[] {
                            Math.Min(colorComponent,1),0, 0});
                    colorComponent += colorDelta;
                }
            }

            // save modified file
            using (Stream outputStream = File.Create(outputFileName))
            {
                doc.Save(outputStream);
            }
        }
    }

    Process.Start(outputFileName);
}
```

You can set stroking or non-stroking colors, examine drawing rule or operation type used, even examine the path or add some content by using AddContent method if you need.

The resulting document produced by the code is shown below:



Pic. 6 Edited graphics paths

Replacing resources in PDF documents

You probably know that PDF documents can contains various resources like fonts, tiling patterns, images, FormXObjects, colorprofiles etc. Whenever you need to replace a resource you can use a special API created for that.

Every FixedDocument (our name for PDF document) has its own resource manager accessible by the property of the same name. So in order to change the resource you can use the following code (relevant part is highlighted):

```
static void Main(string[] args)
{
    using (Stream inputStream = File.Open("../../../data/patternFill.pdf",
       FileMode.Open, FileAccess.Read))
    {
        using (FixedDocument doc = new FixedDocument(inputStream))
        {
            // create a new tiling pattern
            TilingPattern pattern = new TilingPattern("myNewPattern",
                     new Boundary(0, 0, 20, 20), 25, 25);
            pattern.Content.SetDeviceNonStrokingColor(new double[] { 0.1, 0.5, 0.7 });
            pattern.Content.FillAndStrokePath(Path.CreateCircle(10, 10,9));

            // register new pattern as a resource
            doc.ResourceManager.RegisterResource(pattern);

            // replace the old pattern with new one
            doc.ResourceManager.RegisterReplacement("myPattern","myNewPattern");

            //save modified file
            using (Stream outputStream = File.Create(outputFileName))
            {
                doc.Save(outputStream);
            }
        }
    }

    Process.Start(outputFileName);
}
```

In this example we replaced the old tiling pattern resource with the new one. Using this technique you can change the appearance of the PDF documents just by changing resources used by drawing operations.

Summary

In this article we demonstrated a few possible scenarios for content editing, removal and replacement in PDF. The topic is quite extensive, so probably we didn't cover your particular case or maybe you have a specific question. If you need any help with the API or a professional advice just drop us an email, and we'll be happy to assist you. All samples used in this article can be found in our github repo as well.