How to apply clipping mask for drawings on PDF page

Written by Apitron Documentation Team

Introduction

When you work on PDF export feature for your custom CAD application or just want to take advantage of the advanced PDF drawing capabilities you often need the clipping feature. The PDF supports both clipping paths usage and text-based clipping and in this article we'll show you how to use them and apply the clipping while drawing on PDF page. (In addition, there are soft mask and image mask features which are described in our book).

Apitron PDF Kit provides a fixed layout API that supports all possible drawing features defined in PDF standard. When you need to add a drawing, you create a ***ClippedContent*** object that acts as a canvas for your drawing. Its constructor accepts clipping path object that gets set as the clipping mask for your drawing. If you need to combine the current clipping with another, you create an additional clipping content object and add it inside the current one.

Text clipping works a bit different. You can set any text string as a clipping mask by creating a regular *TextObject*, setting its rendering mode to the one that affects clipping, and adding it to page's content. This text then becomes a part of the clipping mask created by *intersecting* it with the current clipping path.

The code

Code sample, provided below, demonstrates regular clipping as well as text-based clipping.

```
// demonstrates how to use text string as clipping path
public static ClippedContent DrawContentUsingTextClipping()
{
    ClippedContent clippedContent = new ClippedContent(0,0,200,200);
    // create text object
    TextObject clipText = new TextObject(StandardFonts.HelveticaBold, 30);
    // set text rendering mode that applies clipping
    clipText.SetTextRenderingMode(RenderingMode.SetAsClipping);
    clipText.AppendText("Text clipping!");

    // set current fill color
    clippedContent.SetDeviceNonStrokingColor(RgbColors.Red.Components);
    // position the text
    clippedContent.Translate(0, 20);
    clippedContent.AppendText(clipText);
    clippedContent.Translate(0, -30);
    // draw image through our clipping
    clippedContent.AppendImage("gradient",0,0,200,200);
    return clippedContent;
}

// demonstrates how to use regular path as clipping path
public static ClippedContent DrawContentUsingClippingPath()
{
    // create base clipping path comprising two circles drawn
    // in different directions one inside another
    Path clippingPath = new Path();
    clippingPath.AppendPath(Path.CreateCircle(150, 600, 100));
    clippingPath.AppendPath(Path.CreateCircle(150, 600, 50, false));
    // create clipped content object and set its clipping path,
    // we also set the clipping rule to even-odd to get the
    // donut-shaped clipping area
    ClippedContent clippedContent = new ClippedContent(clippingPath, FillRule.EvenOdd);
    // set current fill color
    clippedContent.SetDeviceNonStrokingColor(RgbColors.Red.Components);
    // draw rectanle through our clipping
    clippedContent.FillPath(Path.CreateRect(50, 500, 300, 200));
    return clippedContent;
}

static void Main(string[] args)
{
    // create document and register image resource
    FixedDocument doc = new FixedDocument();
    doc.ResourceManager.RegisterResource(new Image("gradient","../../data/gradient.jpg"));
    // create page and append our clipped contents to it
    Page page = new Page();
    page.Content.AppendContent(DrawContentUsingClippingPath());
    page.Content.Translate(250,700);
    page.Content.AppendContent(DrawContentUsingTextClipping());

    // append page to document and save it
    doc.Pages.Add(page);
    using (Stream stream = File.Create("clippedContent.pdf")){ doc.Save(stream); }

    Process.Start("clippedContent.pdf");
}
```

The result looks as follows (the complete code sample can be found in our [github repo]):

Pic. 1 Clipping based on graphics path and text string

Here, we created a donut-shaped clipping path using a path based on two circles drawn in opposite directions and setting an even-odd rule for it. Text string becomes a clipping path when we set its text rendering mode to *SetAsClipping*.

Conclusion

With the Apitron PDF Kit for .NET library you can use advanced PDF graphic features such as clippings, shadings and patterns to create complex drawings demonstrating rich visual effects. Everything that can be drawn using PDF graphics system, can be drawn using fixed layout API provided by our library.