How to extract text blocks from pdf page and create pdf to html conversion tool

Written by Apitron Documentation Team

Introduction

In our first post about PDF text extraction we demonstrated how to extract raw and formatted text from PDF document using Apitron PDF Kit for .NET component. As a part of the latest improvements, we've added new text extraction functionality and are eager to share the results with you.

If you were to extract text from PDF document programmatically before this release, you would have to pick one of the following choices:

- Process text blocks yourself by parsing PDF commands and combining the results into a formatted or raw text data. Frankly speaking, it's the hardest way you might choose and it would require the complete understanding of many text-related PDF aspects.
- Use Apitron PDF Kit API by calling `Page::ExtractText()` with `RawText` or `FormattedText` parameter.

While all these techniques are working fine and do what they're meant for, you may still have a need in some easy and convenient way to analyze the text attributing information e.g. color, original font name, size, etc. without diving too much in PDF specifics. That's why we've implemented new text extraction modes in addition to `Raw` and `Formatted`:

- TaggedText – produces XML output, where each PDF text block becomes wrapped by an xml element containing all available appearance information affecting this block. It also reports *coordinates of each block* in *page space* and therefore gives you unique ability to analyze and format text data in any way you like. Read more on this in separate section further.
- HtmlText – further elaboration of TaggedText, actually this mode is no more than an attempt to lessen the efforts needed for PDF to HTML conversion task which many of developers encounter very often. Described in details in separate section.

Note: in evaluation mode only part of the page text can be extracted and the performance of the text extraction might be slightly affected.

PDF text extraction modes

TaggedText

This mode generates tagged output in XML format. For each PDF page it produces a root *page* element containing PDF text blocks represented by the *textblock* element.

Supported attributes for *page* element are:

- width – width of the page
- height – height of the page

Supported attributes for *textblock* element are:

- left – block left coordinate, in PDF coordinate system (relative to lower left corner)
- bottom – block bottom coordinate, in PDF coordinate system
- width – block width
- height – block height
- fontSize – font size in points
- fontFamily – describes the font used within text block
- letterSpacing – additional letter spacing
- wordSpacing – additional word spacing
- fontStretch – indicates a selection of normal, condensed, or expanded face from a font
- fontWeight – defines boldness of the font
- fontStyle – indicates *italic* font
- nonStrokeColor – fill color, rgb
- strokeColor – stroking color, rgb

All coordinates and dimensions are in *page space* and relative to lower left corner of the PDF page. Text blocks are being produced in the same order they appear on PDF page and without any additional formatting applied. Using this markup one may perform analysis of text layout and perform any context related tasks. Having the information about document nature, it becomes possible to create custom-tailored solutions serving the particular needs of application developer.

Sample code and produced XML are below (page 7 from PDF32000_2008 spec):

```
/// <summary>
/// Extracts tagged text from PDF page at given index.
/// </summary>
/// <param name="pdfDoc">PDF document to extract text from.</param>
/// <param name="pageIndex">Page index.</param>
/// <returns>Extracted text or empty string if page wasn't found or empty.</returns>
public string ExtractTaggedText(FixedDocument pdfDoc, int pageIndex)
{
    if (pdfDoc != null && pageIndex >= 0 && pdfDoc.Pages.Count > pageIndex)
    {
        return pdfDoc.Pages[pageIndex].ExtractText(TextExtractionOptions.TaggedText);
    }

    return string.Empty;
}
```

XML:

```
<page width="595" height="842">
  <textblock left="464.94" bottom="791.91" fontSize="10.98" fontFamily="Helvetica-Bold"
     letterSpacing="0.01" wordSpacing="0" width="93.42" height="13.07">PDF 32000-
     1:2008</textblock>
  <textblock left="70.8" bottom="747.24" fontSize="13.98" fontFamily="Arial" fontWeight="700"
     fontStretch="Normal" letterSpacing="0.01" width="60.21"
     height="19.38">Contents</textblock>
  <textblock left="535.08" bottom="747.3" fontSize="9.96" fontFamily="Arial" fontStretch="Normal"
     letterSpacing="0.03" width="23.4" height="13.26">Page</textblock>
  ...
</page>
```

The first text block is highlighted for demonstration purposes.

HtmlText

This text extraction mode is based on tagged text mode and was designed to provide base implementation of the often needed PDF to HTML conversion task. If you need an additional level of specificity you may implement it using xml output produced by tagged text mode.

`HtmlText` mode produces a `<div>` block containing preformatted text wrapped inside `<pre>` elements. Each of these <pre> elements becomes styled according to text properties of the corresponding PDF text object using the inline style. Location of the element within parent block is being set using relative positioning.

Sample code and produced HTML are below (page 6 from PDF32000_2008 spec):

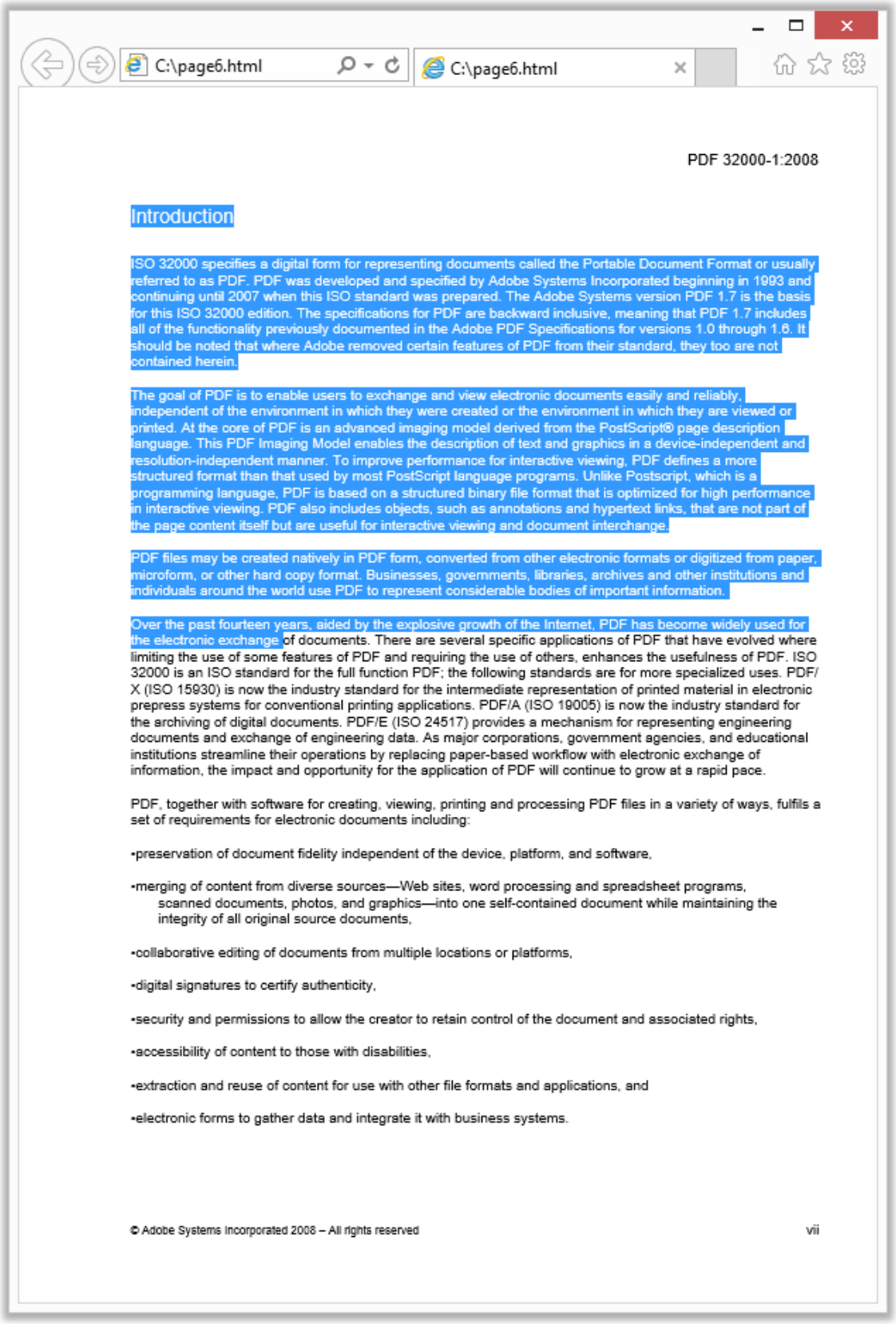
```
/// <summary>
/// Extracts html text from PDF page at given index.
/// </summary>
/// <param name="pdfDoc">PDF document to extract text from.</param>
/// <param name="pageIndex">Page index.</param>
/// <returns>Extracted text or empty string if page wasn't found or empty.</returns>
public string ExtractHtmlText(FixedDocument pdfDoc, int pageIndex)
{
    if (pdfDoc != null && pageIndex >= 0 && pdfDoc.Pages.Count > pageIndex)
    {
        return pdfDoc.Pages[pageIndex].ExtractText(TextExtractionOptions.HtmlText);
    }

    return string.Empty;
}
```

Html:

```
<div style="width:595px;height:842px;position:relative;">
  <style scoped="scoped">pre{margin:0;padding:0;position:absolute;}</style>
  <pre style="left:464.94px;bottom:791.91px;font-size:10.98px;font-
    family:'Helvetica';width:93.42px;height:13.07px;">PDF 32000-1:2008</pre>
  <pre style="left:70.86px;bottom:747.24px;font-size:13.98px;font-
    family:'Helvetica';width:81.62px;height:19.38px;">Introduction</pre>
  <pre style="left:70.86px;bottom:717.42px;font-size:9.96px;font-
    family:'Helvetica';width:490.44px;height:13.26px;">ISO 32000 specifies a digital form for
    representing documents called the Portable Document Format or usually</pre>
  <pre style="left:70.86px;bottom:705.95px;font-size:9.96px;font-
    family:'Helvetica';width:490.37px;height:13.26px;">referred to as PDF. PDF was developed
    and specified by Adobe Systems Incorporated beginning in 1993 and</pre>
  …
</div>
```

All properties of PDF element which can be mapped to html style attribute are used here. Also, as you can see, the produced html block uses scoped style to set the common properties for all nested `<pre>` elements.

If we open the produced html in browser, it will look as follows:



Pic. 1 PDF to HTML conversion results

Conclusion

Using new text extraction modes described in this post, you're now able to perform document text analysis involving its attributes and positions. You can also easily convert PDF to simple html for quick preview purposes.

Apitron PDF Kit for .NET can be used as standard PDF component for creation of applications requiring PDF processing. Create apps for Windows Store, Google Play Store, and Apple Store using same API. Develop server side solutions, cloud, directly managed websites or web services. Being true cross-platform and .NET / Mono / Xamarin compatible, our library raises the implementation of PDF processing logic to the unmatched level.