(/)

HTML to PDF Using OpenPDF

Last modified: October 2, 2021

by baeldung (https://www.baeldung.com/author/baeldung/)

Data (https://www.baeldung.com/category/data/)

HTML (https://www.baeldung.com/tag/html/)

1. Overview

In this quick tutorial, **we'll look at using OpenPDF in Java to convert HTML files to PDF formats programmatically**.

2. OpenPDF

OpenPDF is a free Java library for creating and editing PDF files under the LGPL and MPL licenses. It's a fork of the iText program. In fact, before version 5, the code for generating PDF using OpenPDF was nearly identical to the iText API. It is a well-maintained solution for producing PDFs in Java.

3. Converting Using Flying Saucer

Flying Saucer is a Java library that allows us to render well-formed XML (or XHTML) with CSS 2.1 for style and formatting, generating output to PDF, pictures, and swing panels.

3.1. Maven Dependencies

We'll start with Maven dependencies:

```
<dependency>
    <groupId>org.jsoup</groupId>
    <artifactId>jsoup</artifactId>
    <version>1.13.1</version>
</dependency>
<dependency>
    <groupId>org.xhtmlrenderer</groupId>
    <artifactId>flying-saucer-pdf-openpdf</artifactId>
    <version>9.1.20</version>
</dependency>
```

We'll use the library *jsoup* (https://search.maven.org/classic/#search%7Cgav%7C1%7Cg%3A%22org.jsoup%22%20AND%20a%3A%22jsoup%22) for parsing HTML files, input streams, URLs, and even strings. It offers DOM (Document Object Model) traversal capabilities, CSS, and jQuery-like selectors to extract data from HTML.

The ***flying-saucer-pdf-openpdf*** (https://search.maven.org/classic/#search%7Cgav%7C1%7Cg%3A%22org.xhtmlrenderer%22%20AND%20a%3A%22flying-saucer-pdf-openpdf%22) **library accepts an XML representation of HTML files as input, applies CSS formatting and styling**, and outputs PDF.

3.2. HTML to PDF

In this tutorial, we'll try to cover simple instances that you might encounter in HTML to PDF conversions, such as images in HTML and styling, using Flying Saucer and OpenPDF. We'll also discuss how we can customize the code to accept external styles, images, and fonts.

Let's take a look at our sample HTML code:

```
<html>
    <head>
        <style>
            .center_div {
                border: 1px solid gray;
                margin-left: auto;
                margin-right: auto;
                width: 90%;
                background-color: #d0f0f6;
                text-align: left;
                padding: 8px;
            }
        </style>
        <link href="style.css" rel="stylesheet">
    </head>
    <body>
        <div class="center_div">
            <h1>Hello Baeldung!</h1>
            <img src="Java_logo.png">
            <div class="myclass">
                <p>This is the tutorial to convert html to pdf.</p>
            </div>
        </div>
    </body>
</html>
```

To convert HTML to PDF, we'll first read the HTML file from the defined location:

```
File inputHTML = new File(HTML);
```

As the next step, we'll use *jsoup* (/java-with-jsoup) to convert the above HTML file to a *jsoup Document* to render XHTML.

Given below is the XHTML output:

```
Document document = Jsoup.parse(inputHTML, "UTF-8");
document.outputSettings().syntax(Document.OutputSettings.Syntax.xml);
return document;
```

Now, as the last step, let's create a PDF from the XHTML document we generated in the previous step. The *ITextRenderer* will take this XHTML document and create an output PDF file. Note that **we're wrapping our code in a *try-with-resources (/java-try-with-resources)* block to ensure the output stream is closed**:

```
try (OutputStream outputStream = new FileOutputStream(outputPdf)) {
    ITextRenderer renderer = new ITextRenderer();
    SharedContext sharedContext = renderer.getSharedContext();
    sharedContext.setPrint(true);
    sharedContext.setInteractive(false);
    renderer.setDocumentFromString(xhtml.html());
    renderer.layout();
    renderer.createPDF(outputStream);
}
```

3.3. Customizing for External Styling

We can register additional fonts used in the HTML input document to *ITextRenderer* so that it can include them while generating the PDF:

```
renderer.getFontResolver().addFont(getClass().getClassLoader().getResource("fonts
/PRISTINA.ttf").toString(), true);
```

ITextRenderer may be required to register relative URLs to access the external styles:

```
String baseUrl = FileSystems.getDefault()
  .getPath("src/main/resources/")
  .toUri().toURL().toString();
renderer.setDocumentFromString(xhtml, baseUrl);
```

We can customize image-related attributes by implementing *ReplacedElementFactory*:

```
public ReplacedElement createReplacedElement(LayoutContext lc, BlockBox box,
UserAgentCallback uac, int cssWidth, int cssHeight) {
    Element e = box.getElement();
    String nodeName = e.getNodeName();
    if (nodeName.equals("img")) {
        String imagePath = e.getAttribute("src");
        try {
            InputStream input = new FileInputStream("src/main/resources/"
+imagePath);
            byte[] bytes = IOUtils.toByteArray(input);
            Image image = Image.getInstance(bytes);
            FSImage fsImage = new ITextFSImage(image);
            if (cssWidth != -1 || cssHeight != -1) {
                fsImage.scale(cssWidth, cssHeight);
            } else {
                fsImage.scale(2000, 1000);
            }
            return new ITextImageElement(fsImage);
        } catch (Exception e1) {
            e1.printStackTrace();
        }
    }
    return null;
}
```

Note: The above code prefixes the base path to the image path and sets the default image size in case it isn't provided.

Then, we can add the custom *ReplacedElementFactory* to the *SharedContext*:

```
sharedContext.setReplacedElementFactory(new CustomElementFactoryImpl());
```

4. Converting Using Open HTML

Open HTML to PDF is a Java library that outputs well-formed XML/XHTML (and even some HTML5) to PDF or pictures using CSS 2.1 (and later standards) for layout and formatting.

4.1. Maven Dependencies

In addition to the *jsoup* library shown above, we'll need to add a couple of Open HTML to PDF libraries to our *pom.xml* file:

```
<dependency>
    <groupId>com.openhtmltopdf</groupId>
    <artifactId>openhtmltopdf-core</artifactId>
    <version>1.0.6</version>
</dependency>
<dependency>
    <groupId>com.openhtmltopdf</groupId>
    <artifactId>openhtmltopdf-pdfbox</artifactId>
    <version>1.0.6</version>
</dependency>
```

Library *openhtmltopdf-core* (https://search.maven.org/classic/#search%7Cgav%7C1%7Cg%3A%22com.openhtmltopdf%22%20AND%20a%3A%22openhtmltopdf-core%22) renders well-formed XML/XHTML, and *openhtmltopdf-pdfbox (https://search.maven.org/classic/#search%7Cgav%7C1%7Cg%3A%22com.openhtmltopdf%22%20AND%20a%3A%22openhtmltopdf-pdfbox%22)* generates a PDF document from the rendered representation of the XHTML.

4.2. HTML to PDF

In this program, to convert HTML to PDF using Open HTML, we'll use the same HTML mentioned in section 3.2. We'll first convert the HTML file to a *jsoup Document* as we showed in a previous example.

In the last step, to create a PDF from the XHTML document, ***PdfRendererBuilder* will take this XHTML document and create a PDF as the output file**. Again, we're using *try-with-resources* to wrap our logic:

```
try (OutputStream os = new FileOutputStream(outputPdf)) {
    PdfRendererBuilder builder = new PdfRendererBuilder();
    builder.withUri(outputPdf);
    builder.toStream(os);
    builder.withW3cDocument(new W3CDom().fromJsoup(doc), "/");
    builder.run();
}
```

4.3. Customizing for External Styling

We can register additional fonts used in the HTML input document to *PdfRendererBuilder* so that it can include them with the PDF:

```
builder.useFont(new File(getClass().getClassLoader().getResource("fonts/PRISTINA.
ttf").getFile()), "PRISTINA");
```

PdfRendererBuilder library may also be required to register relative URLs to access the external styles, similar to our earlier example:

```
String baseUrl = FileSystems.getDefault()
  .getPath("src/main/resources/")
  .toUri().toURL().toString();
builder.withW3cDocument(new W3CDom().fromJsoup(doc), baseUrl);
```

5. Conclusion

In this article, we have learned how to convert HTML into PDF using Flying Saucer and Open HTML. We've also discussed how we can register external fonts, styles, and customizations.

As is the custom, all the code samples used in this tutorial are available over on GitHub (https://github.com/eugenp/tutorials/tree/master/pdf).

Comments are closed on this article!

THE COURSES PLATFORM (HTTPS://COURSES.BAELDUNG.COM)

SERIES

JAVA “BACK TO BASICS” TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
HTTPCLIENT 4 TUTORIAL (/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)
SECURITY WITH SPRING (/SECURITY-SPRING)
SPRING REACTIVE TUTORIALS (/SPRING-REACTIVE-GUIDE)

ABOUT

ABOUT BAELDUNG (/ABOUT)
THE FULL ARCHIVE (/FULL_ARCHIVE)
WRITE FOR BAELDUNG (/CONTRIBUTION-GUIDELINES)
EDITORS (/EDITORS)
JOBS (/TAG/ACTIVE-JOB/)
OUR PARTNERS (/PARTNERS)
ADVERTISE ON BAELDUNG (/ADVERTISE)

TERMS OF SERVICE (/TERMS-OF-SERVICE)
PRIVACY POLICY (/PRIVACY-POLICY)
COMPANY INFO (/BAELDUNG-COMPANY-INFO)
CONTACT (/CONTACT)