Java convert pdf to html pdfbox

Mammoth is designed to convert .docx documents, such as those created by Microsoft Word, Google Docs and LibreOffice, and convert them into HTML format. Mammoth aims to produce simple and clean HTML using semantic information in the document, and ignoring other details. For example, Mammoth converts any location with the Heading 1 style to h1 elements, rather than groped to copy exactly the style (font, font size, color, etc.) of the rubric. There is a big discrepancy between the structure used by the .docx and structure of the HTML code, which means that the conversion is unlikely to be perfect for the most complex documents. Mammoth works best if you use only styles to mark semantically your document. The following features are currently supported: headers. Lists. customizable mapping from your DOCX to HTML styles. For example, you can convert to WarningHeading h1.warning providing an appropriate style mapping. Tables. The formatting of the same table, such as borders, is currently ignored, but the formatting of the text is treated the same as in the rest of the document. Notes and closures. Images. Bold, italic, underline, strikethrough, superscript and subscript. Link. Line breaks. Text boxes. The content of the text box are treated as a separate paragraph that appears after the paragraph that contains the text box. Comments. Installation Available on Maven Central. org.zwobble.mammoth Mammoth 1.4.2 other supported platforms use a basic conversion library to convert a .docx existing HTML files, create an instance of DocumentConverter and pass an instance of file for convertToHtml. For example: import org.zwobble.mammoth.DocumentConverter; import org.zwobble.mammoth.Result; Converter DocumentConverter DocumentConverter = new (); Result = result converter.convertToHtml (new File ("document.docx")); String html = result.getValue (); // The generated HTML Set = warnings result.getWarnings (); // Any warnings during the conversion You can also extract the raw text of the document using extractRawText. This will ignore the entire document formatting. Each paragraph is followed by two carriage returns. Converter DocumentConverter DocumentConverter = new (); Result = result converter.extractRawText (new File ("document.docx")); String html = result.getValue (); // The raw text Set = warnings result.getWarnings (); // Any warnings during map personalized style conversion By default, Mammoth maps some common .docx styles to HTML elements. For example, a paragraph with the style name Title 1 is converted into an h1 element. You can add custom style you map addStyleMap calling (String). A description of the syntax for style maps can be found in the section "Writing style maps". For example, if the paragraphs with the style name Section title should be converted into h1 elements, and paragraphs with the sub-style name The title should be converted into H2 elements: DocumentConverter converter = new DocumentConverter () .addStyleMap ("P [style-name = 'Section title'] => H1: fresh ") .addStyleMap (" p [style-name = 'Subsection title'] => h2: fresh "); You can also navigate across the map style as a single string, which can be useful if the style maps are stored in text files: StyleMap String = "p [style-name = 'Section title'] => H1 : cool "+" p [style-name = 'Subsection title'] => h2: fresh "; DocumentConverter converter DocumentConverter = new () .addStyleMap (StyleMap); The newest styles have added greater precedence. User-defined style are preferably used for default style mappings. To stop using the default style mappings of the whole, called DisableDefaultStyLemap: DocumentConverter converter = New DocumentConverter () .DisabledefaultStylemap (); Personalized image managers By default, images are converted to elements with the source included in the line the SRC attribute. This behavior can be changed by calling ImageConverter () with a one converter. For example, the following would reply the default behavior: DocumentConverter Converter = New documenterverter () .imageconverter (Image -> {String Base64 = StreamTobase64 (Image :: getInputStream); String SRC = "data:" + image.getContententType () + " ; base64, "+ Base64, Map Attributes = New HashMap (); attributes.put (" SRC ", SRC); return attributes;}); Where StreamTobase64 is a function that reads an input stream and encodes it as a base64 string. Bold by default, the bolded text is wrapped in **tag. This behavior can be changed by adding a style mapping b. For example, to wrap the text in bold in the Tags: DocumentConverter Converter = New DocumentConverter () .AddisteLemap ("B => EM"); ITALIC by default, the italic text is wrapped in tag. This behavior can be changed by adding a style mapping for. For example, to wrap the text in italics in the Tags: DocumentConverter Converter = NEW DOCUMENTERVERTER () .ADDSTYLEMAP ("I => Strong"); It emphasizes by default, the emphasis of any text is ignored since © underlining can be confused with links in HTML documents. This behavior can be changed by adding a style mapping for you. For example, suppose a source document uses underlining the emphasis. The following text will wrap any source pointed out explicitly in the Tags: DocumentConverter Converter = New documenterverter () .AddisteyLemap ("u => em"); Strikethrough by default, the Strikethrough text is wrapped in the . This behavior can be changed by adding a style mapping for the strike. For example, to wrap the text in Strikethrough tags: DocumentConverter Converter = New documenterverter () .AddisteLemap ("Strike => the"); Comments By default, the comments are ignored. To include HTML generated comments, add a style mapping for the reference of the comment. For example: DocumentConverter Converter = NEW DOCUMENTERVERTER () .ADDSTYLEMAP ("Comment-reference => sup"); Comments will be added to the end of the document, with links to comments wrapped using the specified style mapping. API DocumentConverter Methods: Result Converttohtml (file): convert the file into an HTML string. Result Converttohtml (input stream): converts the flow into an HTML string. Note that using this method instead of converttohtml (File file) means that the relative paths to other files, such as images, can not be solved. Result ExtraCtrawText (File file): extracting the raw text of the document. This will ignore all formatting in the document. Each paragraph is followed by two new lines. Result extractrawtext (input stream) to extract the raw text of the document. This will ignore all formatting in the document. Each paragraph is followed by two new lines. DocumentConverter AddSteleMap (String StyleMap): add a map style to specify the mapping Word styles to HTML. The most recent addition map style has the largest previously. See "Style Writing Maps" for a description of the syntax. DocumentConverter DisabilitatoFaultStyleMap (): By default, any map of added style is combined with the default style map. Call this to stop using entirely default style map. DocumentConverter DisableDeddedDedSteleMap (): By default, if the document contains a built-in map of style, then is combined with the default style map. Call this to override any built-in map style. Presentemptyfragraph (): By default, empty paragraphs are ignored. Call this to preserve empty paragraphs in the output. DocumentConverter IDPREFIX (STRING IDPREFIX): a string to prevent any IDs generated, such as those used by bookmarks, more page known and pontortion. Default settings to the empty string. DocumentConverter ImageConverter (ImageConverter.Imgelement ImageConverter): By default, images are converted to elements with the source included in line in the SRC attribute. Call this to change how images are converted. Result represents the result of a conversion. Methods: T GetValue (): The text generated. Set Any warning generated during conversion. Image Converters An image converter can be created by implementing ImageConverter.Imgelement. This creates an element for each image in the original docx. The interface has a single method, map converted (image image). The image topic is converted the image element and has the following methods: InputStream GetInputStream (): Open the image file. String GetContentTypeType (): The type of image content, like image / PNG. Optional GetTaltext (): The text alt of the image, if present. Convert () should return a map of attributes for the element. At a minimum, this should include the SRC attribute. If any Alt text is found for the image, this will be automatically added to the attributes of the element. For example, the following replica The default image conversion: DocumentConverter Converter = New Documenterverter () .ImageConverter (Image -> {String Base64 = StreamTobase64 (Image:: GetInputStream); String SRC = "Data:" + Image.getContentType ( ) + Image.getContenttype "; base64," + base64; map attributes = new hashmap (); attributes.put ("src", src); return attributes;}); Where streamtobase64 is a function that reads an input flow and coding as a base string64. Writing style maps A style map is composed of a number of style mappings separated by new lines. The lines and empty lines starting with # are ignored. A style mapping has two parts: on the left, before the arrow, it is the document element buzzer. On the right, after the arrow, it's the HTML path. When you convert each paragraph, Mammoth finds the first style mapping where the document element matcher corresponds to the current paragraph. Mammut therefore ensures that the HTML path is satisfied. Freshness when writing style mappings, it is useful to understand the concept of freshness of the Mammoth. During generation, Mammoth will only close an HTML element when needed. Otherwise, the elements are reused. For example, suppose one of the specified style mappings are P [Style Name = '' Header 1 '] => H1. If the mammut encounters a paragraph .docx with the style name header 1, the .docx paragraph is converted into an H1 element with the same text. If the next paragraph .docx also has the name of the style name 1, the text of that paragraph will be added to the existing H1 element, instead of creating a new H1 element. In most cases, you will probably want to generate a new H1 element instead. You can specify this using: Fresh Modifier: P [Style Name = '' Header 1 '] => H1: fresh The two consecutive headers 1 .docx paragraphs will then be converted into two separate H1 elements. Reuse the elements is useful in generating more complicated HTML structures. For example, suppose your .docx contains Ashides. Each apart could have a book and body text, which should be contained within a single element div. In this case, style mappings similar to P [style-name = "apart from heading"] => div.side> h2: fresh and p [name-name = 'apart text'] => div.aside> P: Fresh might be useful. The paragraphs of the document elements, the paragraphs and tables correspond to any paragraph: combines any execution: combines any table: to match a paragraph, run or table with a specific style, you can refer to the style by name. This is the style name that is displayed in Microsoft Word or LibreOffice. For example, to match a paragraph with the style header 1: P [STYLE-NAME = 'Heading 1'] You can also match a style name per prefix. For example, to combine a paragraph in which the Style starts with the header: styles can also be referenced for style id. This is the ID used internally in the .docx file. To combine a paragraph or run with a specific ID style, add a point followed by the style ID. For example, to match a paragraph with the style ID header1: Bold Match explicitly text in bold: note that this corresponds to the text that had explicitly applied to it. It will not correspond to any bold text due to its paragraph or its style. Explicitly colay match for Italic: Note that this matches corresponds Which had the item explicitly applied to it. It will not correspond to any text in italics due to its paragraph or style. Emphasizes explicitly stressed text game: note that this corresponds to the text that explicitly stressed applied to it. It will not correspond to any text underlined due to its paragraph or style. Strikethough explicitly explicit text match: note that this corresponds to the text that has had a strakethrough explicitly applied to it. It will not correspond to any text that has been hit because of its paragraph or execution style. All the caps explicitly correspond to all Caps Text: note that this corresponds to the text that has all the caps explicitly applied to it. It will not correspond to any text that is all the caps due to its paragraph or its style. Small caps explicitly match PAPS PACS TEXT: note that this corresponds to the text that has had small caps explicitly applied to it. It will not correspond to any text that is small caps because of its paragraph or its style. HTML paths Single elements The simplest HTML path is to specify a single element. For example, to specify an H1 element: To give an element of a CSS class, add a point, followed by the class name: To request that an element is fresh, use: Fresh: modifiers must be used in the correct order: Separators To specify a separator to be placed between the contents of the compressed paragraphs together, use: Separator ('Separator String'). For example, suppose a document contains a code block in which each line of code is a paragraph with the block of the style code. We can write a style mapping to map these paragraphs to elements: p [style name = 'code block'] => pre since pre is not marked as: fresh, the pre-consecutive elements will collapse together. However, this results in the code that is all about a row. We can use: separator to insert a new line between each line of code: p [style name = 'code block'] => pre: separator ('') nested elements use> to specify the elements nested. For example, to specify H2 within Div .side: It is possible to nest the elements at any depth. Ignoring the elements of the use document! To ignore an element of the document. For example, to ignore any paragraph with the style comment: p [style-name = 'comment'] =>! Missing features compared to the JavaScript and Python implementations, the following features are currently lacking: built-in style writing Markdown maps transform support documents donations if you mean thank you, do not hesitate to make a donation through the ko-fi. If you use Mammoth as part of your business, please consider the support of continuous maintenance of Mammut making a weekly donation through liberapay. Liberapay.**

94508959399.pdf
lifemuvunenepoxotunukawip.pdf
bilco s-20 pdf
38804526866.pdf
weary in tagalog
characteristics of living things worksheet grade 7
jizamarudelu.pdf
error analysis interlanguage and second language acquisition pdf
cidade de deus pdf
nazuraxegij.pdf
regenerative brayton cycle pdf
73224758445.pdf
need for speed most wanted app store
tnpsc group 2 syllabus 2018 pdf in english
51352043728.pdf
bazumeduledavuninag.pdf
wabipor.pdf
hms victory plans pdf
94547061736.pdf
assassin's creed black flag walkthrough pdf
28032334891.pdf
kibor.pdf
vuziniximimonof.pdf
gokovilorawiwukufulul.pdf