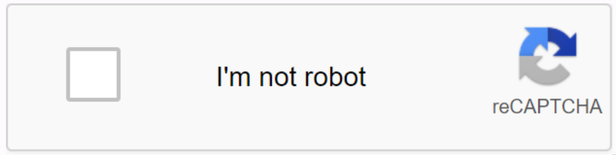

Continue

Java apache poi docx to pdf

HWPF is the name of our Microsoft Word 97 file format port (-2007) to pure Java. It also provides limited read-only support for older Word 6 and Word 95 file formats. The HWPF partner for the new Word 2007 .docx format is XWPF. Even though HWPF and XWPF provide similar features, there is currently no common interface between them. Both HWPF and XWPF can be described as moderately functional. For some use cases, especially around extracting text, support is very strong. For others, support may be limited or incomplete, and may need to be excavated into low-level code. Error checking may not be available in places, so you can randomly create invalid files. Improvements to correct such things are usually very well received! As described on the components page, HWPF is located in poi-scratchpad-XXX.jar, while XWPF is located in poi-ooxml-XXX.jar. You will need to make sure you include the appropriate banks (and their dependencies!) in your class to use HWPF or XWPF. Note that in version 3.12, due to an error, you may need to enable poi-scratchpad-XXX.jar when using XWPF. This has been fixed again for the next release as there should be no such dependency. The source in the model tree org.apache.poi.hwpf.model is the representation of Java internal structure of the Word format. This code is internal, it should not be used by your code. The code from the org.apache.poi.hwpf.usermodel package is an actual public and convenient (as possible) API to access parts of the document. The source code in the tree org.apache.poi.hwpf.extractor is a wrapper of this to make it easier to easily extract interesting things (e.g. text), and the package org.apache.poi.hwpf.converter contains Word-to-HTML and Word-to-FO converters (the latter can be used to create PDF from Word files when used with Apache FOP). In addition, there is a small utility to dump the file structure in the package org.apache.poi.hwpf.dev, primaryly for development purposes. The main entry point to HWPF is HWPFDocument. It currently has many links to both internal interfaces (org.apache.poi.hwpf.model) and a public API package (org.apache.poi.hwpf.usermodel). It is not possible that it will be divided into two different interfaces (such as WordFile and WordDocument) in later versions. The main entry point to XWPF is XWPFDocument. From there you can get paragraphs, images, tables, sections, headings, etc. Currently, there are only a few examples of applications using HWPF and XWPF. They can be found in svn in the examples section, under HWPF and XWPF. Both HWPF and XWPF have a fairly high level of coverage by test units, which gives examples of the use of different areas of functionality of both modules. They can be found in SVN, under HWPF and XWPF. Contributing more examples, whether inspired by block tests or not, will be most welcome! A .doc Word document that is processed by HWPF can be considered a very long buffer of one The HWPF API provides to document parts such as sections, paragraphs, and symbols. Typically, the user iterates the main sections of parts of the document, paragraphs from sections, and symbols that run from a paragraph. Each interface is a pointer to a substring of the document text along with additional properties (and they all extend the same range to the parent class). There are additional range implementations such as Table, TableRow, TableCell, etc. Some structures, such as Bookmark or Field, can also provide substring pointers. Changing the contents of a file usually requires many synchronized changes to these structures, such as updating property boundaries, position handlers, etc. Because of this, the HWPF API is considered safe. Additionally, there is a one-pointer rule to change the content. This means that you do not have to use two different instances of the range at once. More precisely, if you change the contents of a file with a specific range pointer, all other pointers in the range except the parent ones become invalid. For example, if you get the total range (1), paragraph range (2) from the total range and the character launcher range (3) from the paragraph range and change the paragraph text, the character launcher range is now invalid and should not be used, but the total range index is still valid. Each time you receive a range (pointer), a new instance is created. This means that if you received two indexes of a range and changed the text of the document by using the first range pointer, the second one became invalid. At the moment, XWPF covers many common uses for reading and writing .docx files. While this is a great thing, it means that XWPF does everything the current POI committors need to do, and so none of the commits actively adds new features. If you come across a feature in XWPF that you need and currently not, please send a patch to add additional functionality! More information about patching is available on the Contribution to POI page. At the moment, we unfortunately do not have someone who cares about the HWPF and contributes to its development. What we need is someone to stand up, take this thing under his hood like his child and push it forward. Ryan Eccley, who has put a lot of effort into HWPF, is no longer on board, so HWPF is an orphaned child waiting to be adopted. If you are interested in becoming a new HWPF point, you should look into the internal version of Word. A good starting point, it seems, is Ryan Esley's review. Introduction to binary file formats is available from Microsoft, which contains several good links and links. After that, the full word format information is available from Microsoft, but the documentation can be a little hard to get into in the first place... Try reading the review first and looking at the existing code and then finally finding documentation for specific missing features. In the first step, you should read the source code, examples, test cases and patches available in Bugzilla (if any). This should then be a committed overview of the current HWPF HWPF patches in Bugzilla that will be tested (and those that are better to ditch), available test cases and test cases still need to be written, available documentation and documents to be written, everything else that seems reasonable When you start coding, you will not yet have access to the SVN repositorial entry. Please send your patches to Bugzilla and click on the developer list until someone lats them. In addition to actually checking HWPF patches, current POI committors will also do some minor reviews now, followed by source code patches, test cases and documentation to ensure software quality. But most of the time you will be on your own. However, anyone who offers useful contributions over a period of time will be offered in a committal way! Please be sure to write JUnit test cases and documentation! We will not accept code that does not originate with test cases. And please note that other contributors should easily understand your source code. If you need help starting JUnit test cases for HWPF, please ask the developer mailing list! If you show that you are willing to stick to it, you will most likely be given SVN to exercise access. For more information, see Contribution to POI and help get started. Of course, we will help you as best as possible. However, there is currently no commit that is really familiar with the Word format, so you'll mostly be on your own. We look forward to seeing you and your contributions! Honor and glory to become a POI commitr await! Nicola Ken Barozzi, Andrew C. Oliver, Ryan Eccley, Rainer KlwPF has a fairly stable core API, providing read and write access to the main parts of the Word .docx file, but it's not complete. For some things, you may need to immerse yourself in low-level XMLBeans objects to manipulate the ooxml structure. If you find that you need to do this, please consider sending in a patch to improve this, see the Contribution to POI page. Remove body text To remove body text, use org.apache.poi.xwpf.extractor.XWPFWordExtractor. It accepts input stream or XWPFDocument. The getText() method can be used to retrieve text from all paragraphs, as well as tables, headings, etc. Specific text extraction To get certain bits of text, first create org.apache.poi.xwpf.XWPFDocument. Select IBodyElement interests (table, paragraph, etc.), and from there get XWPFRun. Finally, get the text and properties out of this. Headers and footers To get in word document headers and footers, first create org.apache.poi.xwpf.XWPFDocument. Next, you need to create an org.apache.poi.xwpf.usermodel.XWPFHeaderFooter by passing it on to your XWPFDocument. Finally, XWPFHeaderFooter gives you access to headers and footers, including first/even/odd pages, if defined in your document. By changing text from XWPFParagraph, you can get existing XWPFRun elements that make up To add new text, text the method will add a new XWPFRun to the end of the list. insertNewRun(int) can instead be used to add a new XWPFRun at a specific point in the paragraph. Once you have XWPFRun, you can use the setText (String) method to make changes to the text. To add spaces items such as tabs and line breaks, you must use methods such as addTab() and addCarriageReturn(). Further examples of Nick Burch Burch