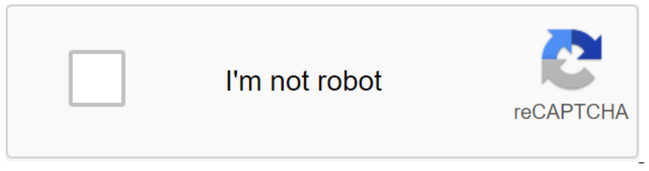

Continue

Apache poi doc to pdf

HWPF is the name of our Microsoft Word 97 (2007) file port to pure Java. It also provides limited reading support for old Word 6 and Word 95 file formats only. The partner of HWPF for the new Word 2007 format .docx is XWPF. Although 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 usage, especially around text extraction, support is very strong. For others, support may be limited or incomplete, and you may need to dig through the low-level code. Checking bugs may not be available in places, so it may be possible to accidentally generate invalid files. Improvements to fix such things are usually very well received! As detailed on the component page, HWPF is found in poi-scratchpad-XXX.jar, while XWPF is in poi-ooxml-XXX.jar. You will need to make sure that you include the appropriate cans (and their dependencies!) in your classpath to use HWPF or XWPF. Please note that in version 3.12, due to an error, you may want to turn on 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 tree org.apache.poi.hwpf.model is a Java view of the internal structure of the Word format. This code is internal and should not be used by your code. The code from the org.apache.poi.hwpf.usermodel package is an actual public and convenient (as far as possible) API to access parts of documents. The source code in the tree org.apache.poi.hwpf.extractor is a wrapper of this to facilitate easy extraction of interesting things (such as text), and the package org.apache.poi.hwpf.converter contains Word-to-HTML converters and Word-to-FO (the latter can be used to generate PDF from Word files when used with Apache FOP). In addition, the package org.apache.poi.hwpf.dev has a small file-structure-dumping utility, primarily for developing purposes. The main entry point for HWPF is HWPFDocument. Currently it has many links to both the internal interfaces (org.apache.poi.hwpf.model package) and the public API (org.apache.poi.hwpf.usermodel) package. It is possible that it will be divided into two different interfaces (such as WordFile and WordDocument) in later versions. The main entry point for XWPF is XWPFDocument. From there you can get paragraphs, photos, tables, sections, blanks, etc. 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 with unit tests, which provides examples of the use of different areas both modules. They can be found in svn, under HWPF and XWPF. The contribution of more examples, whether inspired by a test unit or not, will be most welcomed! A .doc Word how HWPF is processed can be seen as a very long buffer of a single text. The HWPF API provides pointers to document parts such as sections, paragraphs, and character starts. Usually the user will iterize over the main sections of the document, paragraphs from the sections and the symbol runs out of paragraph. Each interface is a pointer for documenting the text subma order along with additional properties (and they all expand the same Parent Range class). There are additional range implementations such as TableRow, TableRow, TableCell, etc. Some structures, such as Bookmark or Field, can also provide subranges. Changing file content typically requires a lot of synchronized changes in these structures, such as updating property boundaries, position handlers, etc. In addition, there is a one-pointer rule for changing content. This means that you don't have to use two different instances of the range at the same time. More precisely, if you change the contents of the file using some range pointers, all other range pointers except the parent list become invalid. For example, if you get a total range (1), a range of paragraphs (2) from the general range and range of character launch (3) from the paragraph range and a change in paragraph text, the range of the character launch is now invalid and should not be used, but the overall range index is still valid. Every time you get a range (pointer) a new instance is created. This means that if you received two range pointers and changed the text of the document using the first-band pointer, the second was invalid. At the moment, XWPF covers many common cases of use for reading and writing .docx files. While this is a great thing, it does mean that XWPF does all that current POI committers need to do it, and so none of the committers actively add new features. If you come across the feature in XWPF that you need and currently not exist, please send a patch to add extra functionality! More information about making patches is available on the PoI Contribution page. At the moment, unfortunately, we do not have someone who cares about HWPF and promotes its development. What we need is someone to stand up, take this thing under the hood like their own baby, and move it forward. Ryan Ackley, who has put a lot of effort into HWPF, is no longer on board, so HWPF is an orphan child waiting to be accepted. If you are interested in becoming the new HWPF current, you should look into Microsoft Word internal. A good starting point seems to be Ryan Ackley's review. Introduction to binary file formats is available from Microsoft, which a few good links and links. After that, full word format information is available from Microsoft, but documentation can be a little hard to get into first... Try reading the review first and look at the existing code, and then finally find documentation for specific missing features. How To the first step is to review the source code, examples, test cases, and HWPF patches available in Bugzilla (if any). Then you have to compile an overview of the current status of HWPF, patches in Bugzilla that need to be registered (and those that are better to opt out), available test cases and test cases yet to be written, available documentation and documents to be written, everything else that seems reasonable When you start coding, you will not yet have written access to the SVN repository. Please send your Bugzilla patches and nag the developer list until someone commits them. Aside from actually checking HWPF patches, current POI committers will also do some minor reviews from time to time of your source patches, test cases and documentation to help ensure the quality of the software. But most of the time you will be on your own. However, anyone who offers helpful contributions over a period of time will be offered committership! Please be sure to write JUnit test cases and documentation! We won't accept code that doesn't come with test cases. And please note that other participants should be able to understand your source code easily. If you need help with JUnit's HWPF test cases, please ask a question on the developer mailing list! If you show that you are willing to stick to this, you will most likely be given access to the SVN commit. For more information and help at the beginning of the work, please visit the POI Contribution page. Of course, we will help you as best we can. However, there is currently no committer that is really familiar with the Word format, so you'll mostly be on your own. We look forward to your contribution! Honor and glory to become a POI committer await! Nicola Ken Barozzi, Andrew C. Oliver, Ryan Ackley, Rainer Klute The purpose of this document is to give a brief overview of the high-level format of HWPF documents. This document is not included in in-depth technical details and is intended only as an add-on to Microsoft Word 97-2007 Binary File Format in free access from Microsoft. The format of the OLE file is not discussed in this document. It is assumed that the reader has working knowledge of the POIFS API. The structure of the Word A Word file consists of the text of the document and the data structures that contain information about the text format. Of course, this is a very simplistic illustration. There are fields and macros and other things that have not been considered. At this stage, HWPF is mainly engaged in formatted text. Reading Word Log Files to read the Word HWPF file is a file information block (FIB). This structure is the entry point for the location and size of the text and document data structure. FIB is located at the beginning of the main The text of the document is also in the main thread. Its original location is given as a and its length is given by bytes FIB.ccpText. These two values are not very useful in getting text because of unicode. Unicode text can be intertwined with ASCII. This brings us to a piece of the table. A piece of the table is used to separate text into non-unicode and unicode pieces. Size and compensation are given in FIB.fcClx and FIB.lcbClx respectively. A piece of the table may contain property modifiers (prm). They are for complex (quickly saved) files and are missed. Each part of the text contains offsets in the main thread that contain the text for that part. If a piece uses unicode, the file offset is masked by a certain bit. Then you have to expose the bit and split into 2 to get the real offset file. Text formatting All text formatting is based on the styles contained in StyleSheet. StyleSheet is a data structure that contains, among other things, descriptions of styles. Each style description may contain a paragraph style and character style, or simply a character style. Each style description is stored in a compressed version of the file. Basically it's deltas from a different style. After all, you have to chain back to zero style, which is an imaginary style with certain implied meanings. Items and styles of Character Item and Character Formating Properties for document text are stored in the file as deltas of some basic style style. Deltas are used to create a complete non-repressive style in memory. The non-depressive styles of paragraphs are represented by the data structure pargraph Properties (PAP). The unpressive character styles are represented by the Character Properties (CHP) data structure. Document text styles are stored in a condensed format in appropriate formatted disks (FKPs). The compressed PAP is called PAPX, and the compressed CHP is CHPX. FKP locations are stored in a cell table. There are separate bin tables for CHPXs and PAPXs. The location and size of the cell tables are stored in FIB. FKP is a 512 byte ole page. It contains the shifting beginning and end of each paragraph/character in the main thread and compressed properties for that interval. The compressed PAPX is based on its basic style in StyleSheet. The compressed CHPX is based on the basic style of the enclosed paragraph in style. Uncompressed styles and other data structures All compressed properties (CHPX, PAPX, SEPX) contain grpprl. Grpprl is an array of sprams. The spray defines the delta from some basic property. There is a table of possible sprams in the Word 97 specification. Each lit is a two-white operand, followed by a parameter. The size of the parameter depends on the litas. Each lit describes an operation that must be performed in a basic style. After each sprm in grpprl performed on the base style you will have style for the item, character run, section, etc etc etc. apache poi doc to pdf. apache poi doc to docx. apache poi doc to html. apache poi doc to xml. convert doc to docx using apache poi. apache poi rtf to doc