# Генерация файлов Word в Apache POI

**8 мин**

**22K**

[Java\*](https://habr.com/ru/hubs/java/)[Apache\*](https://habr.com/ru/hubs/apache/)

Для языка Java (как, впрочем, и для любого другого языка программирования) всё еще не придумали более простого и действенного способа генерации документов docx, чем библиотека Apache POI. В конце нулевых появился сей высокоуровнеый API, позволящий говорить с формируемым документом не на языке разметки XML, а с помощью удобных полей и выводов.  
  
Судя по моим Google-запросам на протяжении более чем года сообщество пользователей этой библиотеки продержалось года этак до 2012, в то время как новые версии библиотеки всё еще появляются на [главной странице проекта](https://poi.apache.org/). Не на все вопросы, касающиеся формирования самого примитивного документа, есть ответы в документации или stackoverflow, не говоря уже о текстах на русском языке. Постараемся компенсировать этот недостаток данных для тех, кому это может понадобиться.

## Основные классы API

[**XWPFDocument**](https://poi.apache.org/apidocs/dev/org/apache/poi/xwpf/usermodel/XWPFDocument.html) — целостное представление Word документа. В нём не только содержится xml-код, интерпретируемый редакторами (Word, LibreOffice), но также содержатся и методы для определения метаданных отображения — набора стилей, сносок и т.п. В этой статье поговорим о первом, так как работа с метаданными не так явно задокументирована, к тому же многие редакторы успешно справляются с отображением документа и без подсказок.  
  
Итак, предположим, у вас на руках есть (ненужный) файл docx. Преобразуем его в файл zip (осторожно, обратное преобразование путем переименования zip -> docx может сделать файл недоступным для вашего редактора(!)), в получившемся архиве откроем папку word, а в ней — файл document.xml. Перед нами xml-представление word-файла, которое также можно было бы получить через Apache POI, с меньшими трудностями.

File file = **new** File("C:/username/document.docx");  
FileInputStream fis = **new** FileInputStream(file.getAbsolutePath());  
XWPFDocument document = **new** XWPFDocument(fis); // Вот и объект описанного нами класса  
**String** documentLine = document.getDocument().toString();

Для того, чтобы поближе познакомиться с содержимым документа, придется вооружиться еще двумя классами API: XWPFParagraph и XWPFTable.  
  
[**XWPFParagraph**](http://poi.apache.org/apidocs/4.1/org/apache/poi/xwpf/usermodel/XWPFParagraph.html) — как следует из названия, представляет собой параграф документа. Расположен он может быть как внутри XWPFDocument,

document.getParagraphs();  
XWPFParagraph lastParagraph = document.createParagraph();

так и внутри таблицы (если точнее — внутри ячейки таблицы, вложенной в ряд таблицы, вложенного непосредственно в таблицу).

document.createTable().createRow().createCell().addParagraph();

Параграф предоставляет изрядный набор информации для вёрстки и размещения текста. Официальная документация на этот счёт достаточно красноречива: отступы слева и справа, сверху и снизу, в том числе и между строками, добавление гиперссылок и границ для параграфа.  
  
[**XWPFTable**](https://poi.apache.org/apidocs/4.0/org/apache/poi/xwpf/usermodel/XWPFTable.html) — класс, олицетворяющий таблицу. Также как и в XWPFParagraph, XWPFTable можно добавлять к самому документу и к ячейке таблицы (создавая, тем самым, таблицу внутри таблицы). Семантика в таком случае чуточку усложняется.

XWPFTable table = document.createTable(); //Здесь всё просто, создаем таблицу в документе и работаем с ней.  
XWPFCell cell = table.createRow().createCell();//Добавим к таблице ряд, к ряду - ячейку, и используем её.  
XWPFTable innerTable = **new** XWPFTable(cell.getCTTc().addNewTbl(), cell, 2, 2); // Воспользуемся конструктором для добавления таблицы - возьмем cell и её внутренние свойства, а так же зададим число рядов и колонок вложенной таблицы  
cell.insertTable(cell.getTables().size(), innerTable);

[**XWPFRun**](https://poi.apache.org/apidocs/dev/org/apache/poi/xwpf/usermodel/XWPFRun.html) — набор данных о выводе текста внутри параграфа. Находится может только внутри параграфа, создается через вызов метода параграфа-родителя:

paragraph.createRun();

Из нескольких «ранов», как я предпочитаю их называть, и состоит целый параграф текста в Word. Каждый «ран» имеет свою настройку шрифта, его цвета и размера, а также стилизации. Через добавление различных «ранов», подчиняющихся разметке параграфа, можно выводить тексты с совершенно разной стилизацией.  
  
Как становится видно из обзора классов, перенос, скажем, css-стиля в документ будет связан с дополнительной сложностью: часть свойств необходимо будет применить к параграфу docx, часть — к объекту класса XWPFRun.  
  
Итак, библиотека легла в External Libraries/jar лежит под рукой, пора творить.  
  
Создадим документ, добавим таблицу 2х2 и параграф.

XWPFDocument document = **new** XWPFDocument();  
XWPFTable table = document.createTable(2, 2);  
XWPFParagraph paragraph = document.createParagraph();  
fillTable(table);  
fillParagraph(paragraph);

Заполним параграф, добавив ран для вывода текста. После перевода строки стилизация параграфа будет потеряна, и в Word новый параграф будет выведен без красной строки.

**void** fillParagraph(XWPFParagraph paragraph) {  
 paragraph.setIndent(20);  
 XWPFRun run = paragraph.createRun();  
 run.setFontSize(12);  
 run.setFontFamily("Times New Roman");  
 run.setText("My text");  
 run.addBreak();  
 run.setText("New line");  
}

Теперь займёмся заполнением таблицы. Мы можем обращаться не только к уже созданным элементам, но и вызвать у сформированной таблицы метод для добавления рядов или колонок.

**void** fillTable(XWPFTable table) {  
XWPFRow firstRow = table.getRows().get(0);  
XWPFRow secondRow = table.getRows().get(1);  
XWPFRow thirdRow = table.createRow();  
fillRow(firstRow);  
}

Опускаемся глубже, на уровень ряда таблицы. Именно в таком порядке предстаёт разбор таблицы в Apache POI — сначала ряды, потом клетки. Напрямую из таблицы можно получить лишь количество колонок в таблице:

table.getColBandSize();

Итак, ряд.

**void** fillRow(XWPFRow row) {  
 List<XWPFTableCell> cellsList = row.getCells();  
 cellsList.forEach(cell -> fillParagraph(cell.createParagraph()));  
}

Оказавшись в ячейке двигаться глубже уже некуда, поэтому можно снова вызвать наш дуболомный метод по заполнению параграфа, предварительно создав его в таблице.  
  
Итак, можно легко уловить суть структуры документа в Word: вкладывай одно в другое и предоставляй доступ (в том числе и к созданию новых экземпляров). К сожалению, далеко не всегда есть возможность получить последний элемент во вложенной коллекции. Чаще всего приходится пользоваться такими вот ухищрениями:

XWPFRun lastRunOfParagraph = paragraph.getRuns(paragraph.getRuns().size() - 1);

Хорошо, с содержимым таблицы разобрались. Что если нам нужно явно уточнить ширину таблицы, а не оставлять её для волной интерпретации редактора?  
  
Для некоторых на первый взгляд числовых значений, например, ширины таблицы, в Apache POI существуют целые классы.

CTTblWidth widthRepr = table.getCTTbl().getTblPr().addNewTblW();  
widthRepr.setType(STTblWidth.DXA);  
widthRepr.setW(BigInteger.valueOf(4000));

С помощью типа укажем, какая именно ширина нам нужна: auto, pct или dxa. В первом случае таблицы займёт всю предоставленную ей ширину, во втором — процент от всей ширины, указанный позже методом setW. В нашем же случае вмешиватеся специальная единица измерения — dxa, равная 1/20 точки.  
  
Классы, подобные CTTblWidth, используются повсеместно: для определения ширины страницы (PgSize), ширины ячейки и др.

#### **Единцы измерения в Apache POI**

В хорошем документе всё выверенно и расчерчено идеально, вплоть до самого пикселя. Возможно, в теории можно сделать всё средствами Apache POI и без углубления в тему единиц измерения, но лучше уделить им внимание сразу, чтобы избежать недопониманий в духе «почему это схлопнулось» и «когда переместил картинку в word на один сантиметр».  
  
О поддержке сантиметров и остальной метрической системы тут остается только мечтать. Это резонно (каждый шрифт уникален, у каждого редактора своя специфика), но дико неудобно. Придется прибегнуть ко множеству конвертаций, если вы хотите задавать отступы (ведь именно в сантиметрах мы привыкли видеть их в word) в сантиметрах. Итак, указав тип измерения dxa для некоторой ширины, как описно в параграфе выше, мы получаем в распоряжение некоторое точное значение, но абсолютно не представляем как им воспользоваться. Для перевода в сантиметры на stackoverflow есть [формула](https://stackoverflow.com/a/18708337). Для всего остального существует класс [Units](https://poi.apache.org/apidocs/dev/org/apache/poi/util/Units.html). В нем определены как методы для перевода единиц измерения, так и сами соотношения между значениями.

#### **Запись готового документа**

Для записи в конечный файл есть удобный метод XWPFDocument — write. На вход принимается поток, в который пойдёт запись.

document.write(**new** FileOutputStream(**new** File("/path/to/file.docx")));

Если готовый документ нужно куда-то передать можно подать в качестве аргумента не File-, а ByteArrayOutputStream.

## Информация об элементе отображения в формате xml

Имея документ, отображающийся корректно в определенном редакторе, полезно было бы узнать как именно представлен необходимый параграф или другой элемент. Для этого определенны специальные методы, возвращающие объекты классов пакета org.openxmlformats.schemas.wordprocessingml.x2006.main. Из названия (wordprocessingml) видно, что данный набор классов используется только для работы с документами word. Например, для xlsx документов есть пакет spreadsheetml, некоторые классы которого очень и очень похожи на классы wordprocessingml, поэтому конвертация между форматами достаточно затруднена.

paragraph.getCTP();  
table.getCTTbl();

Так, пустой параграф будет иметь скромное представление

**<xml-fragment/>**

Пустая таблица покажет больше интересного.

**<xml-fragment** xmlns:main="http://schemas.openxmlformats.org/wordprocessingml/2006/main"**>**  
 **<main:tblPr>**  
 **<main:tblW** main:w="0" main:type="auto"**/>**  
 **<main:tblBorders>**  
 **<main:top** main:val="single"**/>**  
 **<main:left** main:val="single"**/>**  
 **<main:bottom** main:val="single"**/>**  
 **<main:right** main:val="single"**/>**  
 **<main:insideH** main:val="single"**/>**  
 **<main:insideV** main:val="single"**/>**  
 **</main:tblBorders>**  
 **</main:tblPr>**  
 **<main:tr>**  
 **<main:tc>**  
 **<main:p/>**  
 **</main:tc>**  
 **<main:tc>**  
 **<main:p/>**  
 **</main:tc>**  
 **</main:tr>**  
 **<main:tr>**  
 **<main:tc>**  
 **<main:p/>**  
 **</main:tc>**  
 **<main:tc>**  
 **<main:p/>**  
 **</main:tc>**  
 **</main:tr>**  
 **<main:tr>**  
 **<main:tc>**  
 **<main:p/>**  
 **</main:tc>**  
 **<main:tc>**  
 **<main:p/>**  
 **</main:tc>**  
 **</main:tr>**  
**</xml-fragment>**

Что здесь интересного? Свойства tblPr — всевозможные свойства таблицы. Внутри уже описанная ширина таблицы (установлена 0, но свойство «auto» все равно выведет таблицу в приемлимой, автоматической ширине). Также tblBorders — набор информации о границах таблицы. Далее идёт явно выраженное представление внутренностей таблицы. tr — ряд таблицы, внутри вложенны tc. Внутри tc оказался бы набор вложенный параграфов, если бы мы добавили хотя бы один.  
Попробуем пополнить параграф информацией и посмотреть что из этого получится.

XWPFParagraph xwpfParagraph = document.getParagraphs().get(0);  
 xwpfParagraph.setFirstLineIndent(10);  
 XWPFRun run = xwpfParagraph.createRun();  
 run.setFontFamily("Times New Roman");  
 run.setText("New text");

Получаем:

**<xml-fragment** xmlns:main="http://schemas.openxmlformats.org/wordprocessingml/2006/main"**>**  
 **<main:pPr>**  
 **<main:ind** main:firstLine="10"**/>**  
 **</main:pPr>**  
 **<main:r>**  
 **<main:rPr>**  
 **<main:rFonts** main:ascii="Times New Roman" main:hAnsi="Times New Roman" main:cs="Times New Roman" main:eastAsia="Times New Roman"**/>**  
 **</main:rPr>**  
 **<main:t>**New text**</main:t>**  
 **</main:r>**  
**</xml-fragment>**

Здесь ситуация ровно такая же: объект с мета-информацией (в него добавлена информация об отступе красной строки, который мы вложили в коде), а так же само содержимое: там размещается список «ранов». В первый и единственный мы добавили текст и информацию о шрифте. Эта информация также разделилась внутри «рана» — информация о шрифте попала в rPr, сам текст — в элемент t.

#### **Вместо вывода**

Apache POI предоставляет удобный, и, что не менее важно, бесплатный API для работы с документами. В нем непросто добиться единого отображения во всех редакторах (Office Online и LibreOffice обязательно будут выглядеть иначе), есть множество неудобств с единицами измерения, а так же непонятно где и какие свойства в элементах должны находиться. Тем не менее, работа с этими свойствами подчинена логике, а возможность подглядеть в xml не нарушая эту логику делает разработку гораздо более удобной.

**Теги:**

* [java](https://habr.com/ru/search/?target_type=posts&order=relevance&q=%5Bjava%5D)
* [apache poi](https://habr.com/ru/search/?target_type=posts&order=relevance&q=%5Bapache%20poi%5D)
* [word](https://habr.com/ru/search/?target_type=posts&order=relevance&q=%5Bword%5D)
* [docx](https://habr.com/ru/search/?target_type=posts&order=relevance&q=%5Bdocx%5D)

**Хабы:**

* [Java](https://habr.com/ru/hubs/java/)
* [Apache](https://habr.com/ru/hubs/apache/)