Use of tabs in document that needs to generate/convert to PDF

[OpenXML tabs problem during docx2pdf conversion : PDF output Java forum (docx4java.org)](https://www.docx4java.org/forums/pdf-output-f27/exception-and-tabulator-problem-during-docx2pdf-conversion-t1264.html)

|  |
| --- |
|  |
|  |
| **Office Document API Integration Guide** |
| **Version 1.0 dd-mmm-yyyy** |
| Document ID: -ADDS-007 |

**REVISION HISTORY**

| Version | Author | Date | Summary of changes |
| --- | --- | --- | --- |
| 0.1 |  | 24 Jan 2022 | Initial Release |

TABLE OF CONTENT

[1. Getting Started 1](#_Toc94015688)

[1.1. Purpose 1](#_Toc94015689)

[1.2. Java and Library Configuration 1](#_Toc94015690)

[2. Program flow for invoking the Office Document API 2](#_Toc94015691)

[2.1. Template Initialization and Configuration API 2](#_Toc94015692)

[2.2. API to Merge Simple Fields 2](#_Toc94015693)

[2.3. API to Merge Table Fields 3](#_Toc94015694)

[3. Use of Table in Document Template 4](#_Toc94015695)

# Getting Started

## Purpose

This document serves as a developer guide on the use of the Docx4J library to develop a high-level Office Document API for support mail merging.

## Java and Maven Configuration

The office document API requires Java version 11 and above.

# Program flow for invoking the Office Document API

Merge field name defined in the word document template is enclosed by ${}, be it simple merge field or repeatable row level merge fields in a table.

## Template Initialization and Configuration API

Step 1 : Call createTemplate API, passing in the document template number (aka as letter template number). This API will lookup the document registry in the database and pick up the physical file name of the word template. Internally, it then loads the template via the loadTemplate API.

## API to Merge Simple String Fields

Step 1: Define a Hashmap with string as the key and string as the value.

Step 2: Populate the Hashmap with the merge field name and value. There is no need to enclose the merge field name with ${} as the API will take care of that automatically

Step 3: Once all merge field that has been defined in the word document have been filled in, call the setMergeField API passing in the hashMap as the parameter

## API to Merge Table Fields

A document could have one or more tables in a single mail merge template. As MSWord does not have a concept of labelling or naming a table, the only way to identify a table is the first merge field on the 1st row after the header row. As such, the merge field on the first column MUST be unique in field name if there are more than 1 table. In addition, the 1st entry in the merge field Map object must also be the name of the 1st merge field defined in the table template. For example:

In the following hypothetical template, there are 2 tables. The merge field name for 1st column MUST be different. In this illustration, the merge field name for quantity is the same. Technically, the merge field name for the “Name” column can also be the same.

**Sales Item:**

|  |  |  |
| --- | --- | --- |
| No | Name | Quantity |
| **${no}** | ${sales-item} | ${quantity} |

Total: ${total}

**Standard Item:**

|  |  |  |
| --- | --- | --- |
| No | Name | Quantity |
| **${sno}** | ${std-item} | ${quantity} |

Total: ${total}

Step 1: As per step 1 and 2 described under the section API to Merge Simple Fields, for each table row

Step 2: Create a List of Hashmap where the key and value are both of String type.

Step 3:Once all the key and values have been populated for each row, call the setMergeField APi, passing in a logical table name and the Hashmap.

Note that in thal polymorphic setMergeField API call, the first argument “tableName” is a logical name to group the list of rows based on column values in each Hashmap for eventually populating the respective tables. It is not used to determine which of the 2 tables is the target for populating the rows of merge fields

## Create File based Merged Document

## Step 1: After all merge fields have been populated (as described in section 2.2 and 2.3), call the mailMerge API

## Step 2: Call the saveDoc API to save the mail merged document to file.

## Step 3: After step 1, if the template was meant to be repeatable for different set of merge field data, skip step 2 and call the newTemplatePage to automatically create a page break. Perform step 2.2, 2.3 and 24 in sequence

## Step 4: Call the saveDoc API to save the mail merge document to file.

## Text & Object Alignment

Call setParagraphAlignment API to set the alignment based on the following values, which get updated into the internal property “PARAGRAPH\_ALIGNMENT\_OPTION” :

* LEFT\_ALIGN = 0
* RIGHT\_ALIGN = 1
* TAB\_ALIGN = 2

However, note that this API itself does not change the text justification or alignment. There is still a need to call the following code in order to set the paragraph property correctly:

        if (getParagraphAlignment() == RIGHT\_ALIGN)

            para = justifyParagraph(para, JcEnumeration.RIGHT) ;

        else {

            if (getParagraphAlignment() == TAB\_ALIGN) {

                justifyParagraph(para, TEXT\_TAB\_POSITION);

                R.Tab tab = objectFactory.createRTab();

                run.getContent().add(tab)  ;

            }

        }

The above logic is already found in the following API all:

* createParagaph
* createImage
* createPageNumberParagraph

Hence any API that calls these function just need to call the setParagraphAlignment API to set the justification.

## Dedicated Header & Footer

The createrHeader and createFooter are similar in signature. They both accepts a list of object which can either be a string or byte[] array that represent an image. The byte array an be populated by calling the loadImage API.

        byte[] imageByte = loadImage("d:/data/work/java/docx4j/5SIR.jpg") ;

        List<Object> headerList = new ArrayList<Object>() ;

        headerList.add(imageByte) ;

        headerList.add("NCS Pte Ltd    x");

        headerList.add("Singapore 123455") ;

        createHeader(headerList, HdrFtrRef.FIRST) ;

        List<Object> footerList = new ArrayList<Object>() ;

        footerList.add("Making IT happen") ;

        createFooter(footerList, HdrFtrRef.DEFAULT) ;

Internally, both API will iterate and determine the type of object. It call createParagraph API and createImage respectively for string and byte array type.

        HeaderPart headerPart = new HeaderPart();

        headerPart.setPackage(templatePackage);

        for (Object obj : objects) {

                if (obj instanceof java.lang.String) {

                    headerPart.getContent().add(createParagraph((String)obj)) ;

                } else {

                    if (obj instanceof byte[]) {

                        headerPart.getContent().add(createImage((byte[])(obj), headerPart, LOGO\_SIZE));

                    }

                }

        }

It must be stressed that specifically for footer, the page x of y is automatically injected into the footer by default. Internally, the createParagrah and createImage API(s) contains call to to determine the prevailing paragraph alignment setting as expressed in section 2.5. Hence, it is important to call setParagraphAlignment prior to invoking either of these 2 API(s).

The HeaderPart and FooterPart are enshrined in a section part and different type could be defined based on the 2nd parameter passed to the 2 API(s) :

* HdrFtrRef.FIRST
* HdrFtrRef.DEFAULT (2nd page onwards if a type FIRST is defined or even page if a EVEN type has been defined)
* HdrFtrRef.EVEN (even pages only)

## Create Paragraph in different part of the Word document (createObjectParagraph API)

A word document consists of different document parts, where paragraph can be inserted, primarily

* MainDocumentPart
* HeaderPart (instead of calling createHeader API)
* FooterPart (instead of calling createFooter API)

A generic helper API createObjectParagraph is provided to add paragraph to different parts of the document. It takes the following parameters:

* A list of list of object (i.e. List<List<Object>>. The element in the embedded List<Object> can either be null, a string or a byte[] containing an image. It should contain at most 2 elements. The first element, if not null, will be automatically left justified. **The setParagraphAlignment() API call has no bearing on the justifications**. The second element, if not null, will be automatically tab justified. The tab position is determined by the internal variable IMAGE\_TAB\_POSITION. Each list<Object> element in the list represent a new paragraph.
* Document part
* Size of the image in the event that the list of object contains a byte[] image.

This API is design generically in that the 2nd element could be a null, in which case, the inserted paragraph will be left justified. Likewise, if the 1st element is null, inserted paragraph will be based on the value in the 2nd element and will be right justified starting from the IMAGE\_TAB\_POSITION.

Sample code:

        List<List<Object>> list = new ArrayList<>() ;

        List<Object> line = new ArrayList<>();

        line.add("Lingo Systems") ;

        line.add(imageByte) ;

        list.add(line) ;

        imageByte = loadImage("d:/data/work/java/docx4j/merlion.jpg") ;

        line = new ArrayList<>() ;

        line.add(null) ;

        line.add("Integration") ;

        list.add(line) ;

        createObjectParagraph(list, templateMainDocumentPart, 1) ;

Instead of passing in the mainDocumentPart, it is possible to create header and footer by passing in a newly created HeaderPart or FooterPart via the createHdrPar and createFtrPart respectively. Alternatively, if the header or footer already exist, call the getHdrPart or getFtrPart API(s) to retrieve the existing header or footer part. The object will be appended to the existing header and footer part (i.e won’t override the existing content unlike the createHdrPart and createFtrPart API(s)).

## Create Storage based Document

## Create Stream based Merged Document

## Perform Mail Merging

# Use of Table in Document Template

The variable in a table rows needs to defined as the 2nd row immediately after the header row (static text or can be simple merge field).

Field Type

B – Barcode

D – Date

F – Float

M – Bookmark Image

N – Number

P – Picture (ad hoc) or Image

Q – QR code

S – String

T – Table

Z – Signature