



Plutext Pty Ltd

Docx4j Enterprise

V8.4.11 User Manual

Contents

Overview	4
Version 8.4	4
Distributions.....	5
Installation	6
MergeDocx.....	7
INTRODUCTION.....	7
BASIC USAGE	7
Concatenating several entire docx	7
Concatenating parts of several docx.....	7
Inserting in a table cell	10
Resolving altChunk.....	12
Deleting part of a docx.....	13
OpenDoPE	13
MANY DOCUMENTS.....	15
SETTINGS.....	15
Styles	15
Page breaks	16
Controlling Headers and Footers	17
Page Numbering.....	17
Macros	18
Interaction between ODD_PAGE and Page Number restart	19
Bullets & Numbering.....	20
EVENT MONITORING	21
ADVANCED TOPICS.....	21
overrideTableStyleFontSizeAndJustification.....	21
Document Defaults	21
Editing Document Defaults	22
OLE Helper (for docx, pptx, xlsx).....	26
DOCX, PPTX, XLSX SUPPORT	26
REQUIREMENTS/DEPENDENCIES.....	26
OLE CONCEPTS	26
OLE Linking and Embedding in Microsoft Office.....	26
The OpenXML Specification	28

USAGE	30
Overview	30
Linking/embedding in a docx	31
Linking/embedding in a pptx	33
Linking/embedding in a xlsx.....	33
SAMPLE CODE	34
NEW FILE TYPES	35
KNOWN ISSUES/LIMITATIONS	35
Word 2007	35
Powerpoint 2010 x64	35
Office 2010 support for ODF	35
Office 2011 and 2016 for Mac OSX.....	35
Office Online	35
Digital Signatures (for docx, pptx, xlsx).....	36
INTRODUCTION	36
SIGNATURES IN MICROSOFT OFFICE.....	36
SIGNATURE SUPPORT IN OFFICE.....	39
THE SPECIFICATION	40
SPECIFICATION	42
3.1 Core Generation.....	42
3.2 Core Validation.....	42
XAdES	43
USAGE: INVISIBLE SIGNING	44
USAGE: VISIBLE SIGNATURE.....	45
Inserting a signature line.....	46
Signing the document	47
USAGE: SIGNATURE VALIDATION	47
API SUMMARY.....	48
configureSignature.....	48
setVisualSignature.....	48
sign	49
miscellaneous.....	49
LIMITATIONS	50
Visible Signatures	50
Certificate Validation (should a signature be trusted?)	50
X.590 certificate	51

DSA and RSA algorithms.....	51
KNOWN ISSUES WITH OFFICE	51
Windows Explorer	51
Word Online.....	51
Word for Mac.....	51
Word for Android.....	51
TROUBLESHOOTING.....	52
Validation errors	52
MergePptx.....	53
INTRODUCTION.....	53
USAGE	53
Concatenating several entire pptx.....	53
Concatenating parts of several pptx	54
Other SlideRange constructors	54
Deleting part of a pptx	55
SETTINGS.....	55
Sections	55
ThemeTreatment	55
EVENT MONITORING	57
Appendix 1 - Installation	59
Using Maven	Error! Bookmark not defined.
Maven Dependency Notes.....	Error! Bookmark not defined.
Appendix 2 – LOGGING	61
Appendix 3 – .NET environment	62
Introduction	62
.NET sample solution	62
Logging	62
GAC.....	63
ASP.NET notes.....	63
Recreating the DLLs.....	64

Overview

This manual describes features of the docx4j Enterprise Edition.

The Enterprise Edition includes the following features which are not available in the Community Edition (ie the docx4j open source project):

Docx	MergeDocx	MergeDocx is for appending/concatenating docx files together to create a single docx file
	OLE	<p>OLE Helper makes it easy to use docx4j to programmatically embed or link files (eg PDF or HTML files) as OLE objects in a docx, pptx, or xlsx.</p> <p>Without this OLE Helper, it can be a real challenge to convert your file into a suitably structured OLE object, which works across Office 2007, 2010, 2013 and 2016.</p>
	DigSig	<p>Dig Sig makes it easy to use docx4j to work with digital signatures:</p> <ul style="list-style-type: none">• programmatically sign docx, pptx, or xlsx files<ul style="list-style-type: none">○ with 1 or more signatures○ with or without the signature being “visible” (docx only)• validate signatures
Pptx	MergePptx	MergePptx is a utility for concatenating pptx presentations together. It uses the slide masters and layouts from the first presentation, so it "re-brands" subsequent presentations to have the same look and feel.
	OLE	see above
	DigSig	see above
Xlsx	OLE	see above
	DigSig	see above

Additional exclusive features will be added over time.

Purchasers of the Enterprise Edition also support Plutext’s continued investment in the development and support of the docx4j open source project.

Version 8.4

docx4j 8.0.0 community edition introduced the following changes:

- the minimum supported version of Java is Java 8
- you specify your preferred JAXB implementation by adding one and only one of the following to your project:

```
<!-- use the JAXB shipped in Java 8 -->
<dependency>
  <groupId>org.docx4j</groupId>
  <artifactId>docx4j-JAXB-Internal</artifactId>
  <version>8.3.11</version>
```

```

</dependency>

<!-- use the JAXB Reference Implementation -->
<dependency>
  <groupId>org.docx4j</groupId>
  <artifactId>docx4j-JAXB-ReferenceImpl</artifactId>
  <version>8.3.11</version>
</dependency>

<!-- use the MOXy JAXB implementation -->
<dependency>
  <groupId>org.docx4j</groupId>
  <artifactId>docx4j-JAXB-MOXy</artifactId>
  <version>8.3.11</version>
</dependency>

```

If you are using Maven, including one of the above dependencies will pull in the rest of docx4j automatically.

- docx4j is now a Maven multi-module project, and the sample code is contained in separate modules.

Enterprise Edition v8.1.0 onwards is packaged according to the same principles. Feature-wise:

- MergeDocx: there are some layout improvements
- MergePptx: support for sections
- DigSig and OLE: no changes

In the distribution, there is a sub-directory corresponding to each of the above projects.

In each sub-directory, you'll find the run-time jar, a jar containing sample files, and (if applicable) a jar containing source code.

In the **samples** jar META-INF dir, you'll find a **maven pom.xml**. In this, a dependency on the current release of docx4j (v8.3.11) is declared.

Distributions

The Enterprise Edition is available in 3 flavours for 2 platforms:

Java	time limited trial/evaluation
	product – binaries only
	product – with Java source code
.NET	time limited trial/evaluation
	product – binaries, plus .NET wrapper code
	product – with Java source code (and instructions for creating DLL)

Installation

For Maven-based installation instructions, please see Appendix 1 below.

Otherwise, to use the Enterprise Edition:

- first make sure you have **docx4j 8.3.11** or later running properly in your project, then
- you simply add the relevant jars to your project (and any dependencies provided in the lib dir) depending on whether you want to use MergeDocx, MergePptx, OLE or DigSig components .

The Enterprise Edition DLL is intended for use in a .NET environment. Instructions for use in a .NET environment are contained in appendix 2.

MergeDocx

INTRODUCTION

This chapter explains how to use the MergeDocx functionality, which is capable of appending/concatenating docx files together to create a single docx file. For example, to place a cover letter and a contract into a single docx file, without changing the look/feel of either document.

BASIC USAGE

Concatenating several entire docx

A `BlockRange` is essentially a `WordprocessingMLPackage`, or a range of content in a `WordprocessingMLPackage`, plus config settings.

To merge docx files, you invoke `DocumentBuilder` with `List<BlockRange>`:

```
List<BlockRange> blockRanges = new ArrayList<BlockRange>();
blockRanges.add( new BlockRange( wordMLPkg1 ) );
blockRanges.add( new BlockRange( wordMLPkg2 ) );
// etc

// Perform the actual merge
DocumentBuilder documentBuilder = new DocumentBuilder();
WordprocessingMLPackage output = documentBuilder.buildOpenDocument(blockRanges);
```

You can fine tune the merge process by configuring individual block ranges, or the `DocumentBuilder` object, as described in the `SETTINGS` section below.

The samples directory contains an example called `MergeWholeDocumentsUsingBlockRange` which you can use as a starting point.

Alternatively, there is a [webapp which can generate code](#) for you, based on your chosen configuration.

Note: there is also a static method you can use to merge a `List<WordprocessingMLPackage>`, but that is not recommended since it precludes user config of `DocumentBuilder` and individual `BlockRanges`.

If you invoke `DocumentBuilder` with `List<BlockRange>`, obviously all your `BlockRanges` are in memory at once. `DocumentBuilderIncremental` is a more memory efficient approach which avoids this. See `MANY DOCUMENTS` further below.

Concatenating parts of several docx

If you wish to use only a certain part of the documents, you need to invoke `DocumentBuilder` with a `List<BlockRange>`

`BlockRange` associates a range with a `WordprocessingMLPackage`.

The `org.docx4j.wml.Body` element has a method:

```
public List<Object> getEGBlockLevelElts()
```


which contains the "block-level" document content (paragraphs, tables etc).¹

BlockRange constructors let you say you want the contents starting from the nth element onwards:

```
/**
 * Specify the source package, from "n" (0-based index) to the end of the document */
public BlockRange(WordprocessingMLPackage wordmlPkg, int n)
```

or count elements from the nth element:

```
/**
 * Specify the source package, from "n" (0-based index) and include "count"
 * block-level (paragraph, table etc) elements. */
public BlockRange(WordprocessingMLPackage wordmlPkg, int n, int count)
```

or the entire docx:

```
/**
 * Specify the entire source package. */
public BlockRange(WordprocessingMLPackage wordmlPkg)
```

For example:

```
List<BlockRange> blockRanges = new ArrayList<BlockRange>();
blockRanges.add(new BlockRange(wmlPkgIn)); // add all
blockRanges.add(new BlockRange(wmlPkgIn, 0, 6)); // paras 0-5
blockRanges.add(new BlockRange(wmlPkgIn, 6)); // paras 6 onwards

DocumentBuilder documentBuilder = new DocumentBuilder();
WordprocessingMLPackage output =
    documentBuilder.buildOpenDocument(blockRanges);
```

The result is a new `WordprocessingMLPackage` containing the specified portions of the source documents.

The samples directory contains an example called `MergeBlockRangeFixedN`.

Where you want to use the nth element constructors, how do you determine n? See [Determining the nth element](#) towards the end of this document.

You may use the one `WordprocessingMLPackage` in more than one `BlockRange`. For example:

```
List<BlockRange> blockRanges = new ArrayList<BlockRange>();
blockRanges.add(new BlockRange(wmlpkg1, 12));
blockRanges.add(new BlockRange(wmlpkg2, 3, 3));
blockRanges.add(new BlockRange(wmlpkg1)); // Use wmlpkg1 again
```

You must **not** however, use a `BlockRange` object twice. For example, the following is an incorrect usage:

```
BlockRange blockRange1 = new BlockRange(wmlpkg1, 12);
List<BlockRange> blockRanges = new ArrayList<BlockRange>();
blockRanges.add(blockRange1);
```

¹ Since docx4j 2.7.0, you can also use the `ContentAccessor` interface (which is supported by various objects):

```
public List<Object> getContent()
```

```
blockRanges.add(new BlockRange(wmlpkg2, 3, 3));
blockRanges.add(blockRange1); // Incorrect
```

Determining the nth element

As explained above, BlockRange constructors let you say you want the contents starting from the **nth element onwards**:

```
/**
 * Specify the source package, from "n" (0-based index) to the end of the document */
public BlockRange(WordprocessingMLPackage wordmlPkg, int n)
```

or **count elements from the nth element**:

```
/**
 * Specify the source package, from "n" (0-based index) and include "count"
 * block-level (paragraph, table etc) elements. */
public BlockRange(WordprocessingMLPackage wordmlPkg, int n, int count)
```

The question arises as to how to work out these numbers.

There are three approaches for finding the relevant block:

- manually
- via XPath
- via TraversalUtils

TraversalUtils is the recommended approach. This is mainly because there is a limitation to using XPath in JAXB (as to which see below).

Explanations of the three approaches follow.

Common to all of them however, is the question of how to identify what you are looking for.

- Paragraphs don't have ID's, so you might search for a particular string.
- Or you might search for the first paragraph following a section break.
- A good approach is to use content controls (which can have ID's), and to search for your content control by ID, title or tag.

The examples provided show how to do each of these. They can be readily adapted for other cases, such as before or after a table or image. If you have any difficulties with your particular case, please do not hesitate to ask for support.

Manual approach

The manual approach is to iterate through the block level elements in the document yourself, looking for the paragraph or table or content control which matches your criteria. To do this, you'd use org.docx4j.wml.Body element method:

```
public List<Object> getEGBlockLevelElts()
```

XPath approach

Underlying this approach is the use of XPath to select JAXB nodes:

```
MainDocumentPart documentPart = wordMLPackage.getMainDocumentPart();
String xpath = "//w:p";
List<Object> list = documentPart.getJAXBNodesViaXPath(xpath, false);
```

You then find the index of the returned node in `EGBlockLevelElts`.

Beware, there is a limitation to using XPath in JAXB: the xpath expressions are evaluated against the XML document as it was when first opened in docx4j. You can update the associated XML document **once only**, by passing true into `getJAXBNodesViaXPath`. Updating it again (with current JAXB 2.1.x or 2.2.x) will cause an error. So you need to be a bit careful!

TraversalUtils approach

TraversalUtil is a general approach for traversing the JAXB object tree in the main document part. TraversalUtil has an **interface** `Callback`, which you use to specify how you want to traverse the nodes, and what you want to do to them.

TraversalUtil can be used to find a node; you then get the index of the returned node in `EGBlockLevelElts`.

Examples are in the samples directory, named as follows:

	Manually	via XPath	via TraversalUtil
String	MergeBlockRangeN ViaManualString	MergeBlockRangeN ViaXPathString	MergeBlockRangeN ViaTraversalUtils String
SectPr	MergeBlockRangeN ViaManualSectPr	MergeBlockRangeN ViaXPathSectPr	MergeBlockRangeN ViaTraversalUtils SectPr
Content control	MergeBlockRangeN ViaManualContentControl	MergeBlockRangeN ViaXPathContentControl	MergeBlockRangeN ViaTraversalUtils ContentControl

Inserting in a table cell

The approach described above doesn't allow you to insert contents into a table cell.

To do this, you can either use the class `ProcessAltChunk` as described next page below, or you can use a placeholder to indicate where you want a `BlockRange` to be inserted.

The placeholder is a content control containing a

```
<w:tag w:val="MergeDocx:BlockRangeIDREF=myTableContent"/>
```

in this case referencing a `BlockRange` having ID "myTableContent".

Inside a table cell, the complete placeholder would look something like this:

```
<w:tc>
  <w:sdt>
    <w:sdtPr>
      <w:tag w:val="MergeDocx:BlockRangeIDREF=myTableContent"/>
    
```

```

</w:sdtPr>
<w:sdtContent>
  <w:p>
    <w:r>
      <w:t>My placeholder7</w:t>
    </w:r>
  </w:p>
</w:sdtContent>
</w:sdt>
</w:tc>

```

The BlockRange which will be placed at this location, is given a matching ID:

```
blockrange.setID("myTableContent");
```

You may then invoke DocumentBuilder in the usual way. The result will be that the contents of the table cell are replaced with the contents of the block range.

This works in a similar way to the way AltChunk processing works (see next page); in both cases you can insert the block range at locations where block/paragraph-level content is allowed.

For the best practice approach, please see the end of this section. The interim content works up to that by describing alternatives.

The simplest approach is to add your ID'd block ranges to the blockRanges list *before* the 'real' documents:

```

List<BlockRange> blockRanges = new ArrayList<BlockRange>();
BlockRange block;

// Define insertions
block = new BlockRange(insertionDocx,1,1);
block.setID("MySourceId");
blockRanges.add( block );

// Now add inputDocx1 proper
block = new BlockRange(inputDocx1);
blockRanges.add( block );

// Perform the actual merge

```

The reason for this is that if instead the block range(s) being moved is/are last, then after it is/they are moved, the sectPr at the end of the previous block range is left untouched, and is now adjacent to the document level sectPr. (The step of moving things around is the very last step in the MergeDocx process).

The downside of having your ID'd block ranges at the start of the blockRanges list, is that certain document wide defaults come from there.

If you have them at the end of the blockRanges list, you'll get two sectPr elements at the end of the document (the first belonging to the immediately prior block range, and the document level one an artifact from the block range which was moved). For example:

```

<w:p>
  <w:pPr>
    <w:sectPr w:rsidR="00D37ADB">
      <w:pgSz w:h="16838" w:w="11906"/>
      <w:pgMar w:gutter="0" w:footer="708" w:header="708" w:left="1440" w:right="1440" w:bottom="1440" w:top="1440"/>
      <w:cols w:space="708"/>
      <w:docGrid w:linePitch="360"/>
    </w:sectPr>
  </w:pPr>
</w:p>

```

```

        </w:pPr>
    </w:p>
    <w:sectPr>
        <w:pgSz w:h="16838" w:w="11906"/>
        <w:pgMar w:gutter="0" w:footer="708" w:header="708" w:left="1440" w:right="1440" w:top="1440" w:bottom="1440"/>
        <w:cols w:space="708"/>
        <w:docGrid w:linePitch="360"/>
    </w:sectPr>

```

This is harmless enough, but if you wanted to fix it, you could in your own code programmatically delete the document level one, and you could also promote the sectPr from the last paragraph.

You wouldn't want to setSectionBreakBefore(SectionBreakBefore.NONE) on the block range not being moved, since although you'll end up with only one sectPr, it is the wrong one!

Finally, here is the best practise:

- add your documents
- followed by the fragments to be moved
- ***followed by an empty block range*** constructed from the last document not being moved

like so:

```

WordprocessingMLPackage pkg1 = WordprocessingMLPackage.Load(new File(file1));
BlockRange source1 = new BlockRange(pkg1);
source1.setSectionBreakBefore(SectionBreakBefore.NONE); // note this

BlockRange tableContent = new BlockRange(WordprocessingMLPackage.Load(new
File(file2)));
tableContent.setID("myTableContent");

List<BlockRange> sources = new ArrayList<BlockRange>();
sources.add(source1);
sources.add(tableContent);

// Add pkg1 again for our body level sectPr
BlockRange emptyBR = new BlockRange(pkg1, 0, 0); // none of the contents - just
sectPr
sources.add(emptyBR);

```

Resolving altChunk

altChunk is a way of telling a consuming application that certain content is to be included in the document.

For further details, please see <http://blogs.msdn.com/b/ericwhite/archive/2008/10/27/how-to-use-altchunk-for-document-assembly.aspx>

Word 2007 understands what to do with an altChunk.

docx4j doesn't, unless you use the MergeDocx utility (or write your own code). If your docx contains altChunks, it is important to be able to resolve them if you want to generate HTML or PDF output using docx4j.

MergeDocx handles altChunk of type docx, as opposed to html or plain text. Support for altChunk of type xhtml is available in the docx4j-ImportXHTML jar.

The class ProcessAltChunk is used as follows:

```

ProcessAltChunk processor = new ProcessAltChunk(wordMLPackage);
wordMLPackage = processor.process();

```

In the constructor, you pass the package containing the altChunks you wish to process.

The process() method returns a new package, in which the altChunks have been converted.

There are 2 fields you can set to configure behaviour:

- **setStyleHandler**; this defaults to StyleHandler.USE_EARLIER, which you'd use if all your chunks use the same style definitions. Otherwise if the style definitions are different, use StyleHandler.RENAME_RETAIN
- **setUseContinuousSections**; introduced in 8.1.0.3, the intent of this is to be able to honour margin settings in an altChunk. It does this by inserting continuous section breaks before and after the altChunk, specifying appropriate margins. This capability is not available in Microsoft Word's altChunk conversion.

Limitations/recommendations:

- We recommend you avoid setting headers/footers in your altChunk.

Microsoft Word does strange things when an altChunk contains headers/footers; currently, MergeDocx does not attempt to duplicate this behaviour.

- altChunk elements in parts other than the Main Document Part (eg headers/footers, footnotes/endnotes and comments) are not converted.

Any comments and footnotes/endnotes in the altChunk should get added OK.

Deleting part of a docx

If you want to delete part of a docx, including the parts it references but which will no longer be used, you can use the constructor:

```
/**
 * Specify the source package, from "n" (0-based index) and include "count"
 * block-level (paragraph, table etc) elements. */
public BlockRange(WordprocessingMLPackage wordmlPkg, int n, int count)
```

twice on the one input document, adding the bit before the stuff to be deleted, and the bit after it.

OpenDoPE

Background

The Open XML specification includes a technology called “Custom XML data binding”, which can be used in document automation and reporting scenarios to automatically inject data from an XML document of your choosing into your docx.

If a content control has an XPath, that XPath is used to retrieve the matching element from your XML document.

OpenDoPE (Open Document Processing Ecosystem) is a set of conventions for tagging a content control to enable:

- conditional content
- repeating content (eg rows of a table, or a bulleted or numbered list)

docx4j is the reference implementation of OpenDoPE.

MergeDocx support for OpenDoPE

You can use MergeDocx and OpenDoPE together. Support for combining these technologies was significantly improved in MergeDocx v1.5.0

You can use MergeDocx first, and then docx4j's OpenDoPEHandler.

Or you can use docx4j's OpenDoPEHandler first, then MergeDocx.

Either order is supported, but it is probably more efficient to use MergeDocx first, followed by OpenDoPEHandler. If you plan use MergeDocx first, and your documents include compound conditions (ie and|or|not operators), you must use docx4j 3.0.

MergeDocx is designed to ensure that each of the input docx uses its own OpenDoPE parts and XML answers, without interfering with the other input docx.

There are two approaches to supplying the XML answer files.

The first approach is to inject an appropriate answer file into each input docx before invoking MergeDocx and OpenDoPEHandler. This is the approach which would be familiar to OpenDoPE users.

A second approach is to tell MergeDocx a Map of W3C DOM Documents containing answers which are to be used across the input documents. The map is keyed by root element QName. With this approach, you can skip the preliminary step of injecting real XML data into each input docx.

For example, suppose you were merging 3 documents, of which 2 used an answer file with root element <supplier> and one used an answer file with root element <specification>.

With:

```
Map<QName, org.w3c.dom.Document> answerDomDocs
```

you can set:

```
documentBuilder.setOpenDoPEAnswers(answerDomDocs);
```

and the values supplied will be used in preference to whatever XML part (with corresponding root element QName) is in the input docx.

There is a helper class OpenDoPeRegistration, which adds an InputStream representation of your XML, to the Map<QName, org.w3c.dom.Document>.

```
Map<QName, org.w3c.dom.Document> answerDomDocs = new HashMap<QName, org.w3c.dom.Document>();
InputStream is = FileUtils.openInputStream(new File("supplier.xml"));
OpenDoPeRegistration.register(answerDomDocs, is);
is = FileUtils.openInputStream(new File("specification.xml"));
OpenDoPeRegistration.register(answerDomDocs, is);

documentBuilder.setOpenDoPEAnswers(answerDomDocs);
```

OpenDoPE processing of rich text fragments

OpenDoPE also allows you to bind to an XML node containing:

- escaped XHTML

- docx content (stored as escaped Flat OPC XML) (since docx4j 3.0.1)

In both cases, docx4j will convert that to docx content.

In the Flat OPC XML case, it converts it to an AltChunk (see previous section). MergeDocx can then convert the AltChunk to native document content.

MANY DOCUMENTS

If you invoke `DocumentBuilder` with `List<BlockRange>`, obviously all your `BlockRanges` are in memory at once.

If you are merging many documents, or even a smaller number of large documents, you may run out of memory.

`DocumentBuilderIncremental` is intended to help in this situation. It allows you to work with a single `BlockRange` at a time.

Example of usage:

```
DocumentBuilderIncremental dbi = new DocumentBuilderIncremental();

for (int i = 0; i < MAX; i++) {

    BlockRange block = getBlockRange(i); // Your method

    block.setSectionBreakBefore(BlockRange.SectionBreakBefore.NEXT_PAGE);
    if (i==0) {
        block.setHeaderBehaviour(BlockRange.HfBehaviour.DEFAULT);
        block.setFooterBehaviour(BlockRange.HfBehaviour.DEFAULT);
    } else {
        // Avoid creating unnecessary additional header/footer parts
        block.setHeaderBehaviour(BlockRange.HfBehaviour.INHERIT);
        block.setFooterBehaviour(BlockRange.HfBehaviour.INHERIT);
    }

    System.out.println(i);
    dbi.addBlockRange(block, i==(MAX-1) ); // 2nd param is whether this is your last docx
}

WordprocessingMLPackage output = dbi.finish();// Get the output docx
```

In the example above, the headers/footers are taken from the first document only. This avoids creating potentially thousands of header/footer parts, where just a couple suffice.

SETTINGS

Styles

By default, if a style is encountered which is already defined in an earlier `BlockRange`, that earlier definition will be used. If the definition is different, this will cause the appearance of text using this style to change.

This default option is `USE_EARLIER`. This is similar to the default option in Word when you paste content from one document to another (that is, if there is a style defined in the destination which has the same name (name, *not* ID), then the “destination” style is used).

If the documents you are merging were styled independently, you will probably want them to retain their individual look. This can be accomplished by importing the styles (and renaming them so they don't collide).

To do this, `setStyleHandler` to `RENAME_RETAIN`:

```
BlockRange blockRange1 = ...
BlockRange blockRange2 = ...

source2.setStyleHandler (StyleHandler.RENAME_RETAIN);
```

This is similar to Word's "source formatting", the difference being that in Word, the effect is achieved by using direct paragraph formatting. In contrast, MergeDocx defines and imports a style. This creates less markup, and makes it easier for someone to edit the merged document (ie to add paragraphs using the same style).

Known limitation regarding **Table of Contents**: consider a style which will be renamed. A TOC field which refers to that style will not be updated to use the new name. This means entries in the table of contents will go missing.

If the document contains numbering, you'll also want to :

```
source2.setNumberingHandler (NumberingHandler.ADD_NEW_LIST);
```

Page breaks

MergeDocx ensures that each document is separated by a section properties element. The relevant properties are actually contained in the first `sectPr` element in the second of any two `BlockRanges`.

In other words, if 3 documents are concatenated, and each is just a single section, the resulting document will contain 3 sections.

By default each section starts on a new page.

If you want to avoid the page break, use `BlockRange`'s `setSectionBreakBefore` method:

```
BlockRange blockRange1 = ...
BlockRange blockRange2 = ...

// avoid page break
blockRange2.setSectionBreakBefore (SectionBreakBefore.CONTINUOUS);

etc.
```

The `MergeBlockRangeFixedN` sample utilizes this.

Your choices for the `SectionBreakBefore` property are:

- NONE
- NONE_MERGE_PARAGRAPH
- NEXT_PAGE
- NEXT_COLUMN
- CONTINUOUS
- EVEN_PAGE
- ODD_PAGE

With the exception of "NONE" and "NONE_MERGE_PARAGRAPH" these mirror values available in Word.

Since what happens between documents is controlled by the first sectPr in the second of the two documents, MergeDocx will set the first sectPr in the second document with the value specified. If there is no sectPr, it will add one at the end of the BlockRange and set that.

"NONE" is a bit different. In this case, no sectPr will be added, and nor will any existing sectPr be altered. So you can think of it as "unspecified". NONE can be useful if you want to manipulate sectPr values in your own code.

"NONE_MERGE_PARAGRAPH" will attempt to merge the last paragraph of the previous block range with the first paragraph of this one.

If you leave the property unset, MergeDocx will add a sectPr if one is not present. MergeDocx will not set its type. If the type is not set, the default is NEXT PAGE, according to the OpenXML spec.

Note: in Word, by default, ODD_PAGE is not honoured if you have set page numbering to restart. Please see the section after Page Numbering below for details as to how to control this behaviour.

Also, Word will ignore a "continuous" setting, and insert a page break, if it detects that the page sizes of the two contiguous sections are different. This can produce unexpected results where, for example, both page sizes are intended to be A4 portrait, but specified in units which differ (for whatever reason) by a few mm. The sample **NormalizePageSizes** contains code which demonstrates how to address this issue.

Controlling Headers and Footers

Suppose you are merging docx1 and docx2.

The default behaviour is as follows:

- If docx1 has a header, and docx2 does as well, then by default both sets of headers will be used.
- If docx1 has a header, but docx2 doesn't, then by default the pages from docx2 will be shown using headers from docx1.

You can override this behaviour:

- if you want no headers defined in the first section of docx2:

```
blockRange2.setHeaderBehaviour(HfBehaviour.NONE);
```

- if docx2 has headers defined in its first sectPr, but you want to ignore them and use the headers from docx1:

```
blockRange2.setHeaderBehaviour(HfBehaviour.INHERIT);
```

There is a similar method for controlling footer behaviour, called setfooterBehaviour.

Page Numbering

Suppose you are merging docx1 and docx2, and showing page numbers or cross referencing to page numbers.

Unless docx2 explicitly restarts page numbering, the numbers will continue on from those in docx1.

You can make the page numbering restart with:

```
blockRange2.setRestartPageNumbering(true);
```

If you are using page numbering of the form "page n of <total pages>" and you want <total pages> to reflect the number of pages in the relevant original document (rather than the number of pages in the resulting merged document), you should change your source documents so that they refer to <Total Number of Pages in Section>. See further <http://support.microsoft.com/kb/191029>

This will work provided each source docx has a single section. If the source documents have multiple sections, you will need to put a bookmark on the last page of each, and use a reference to that as the total number of pages.

If you have front matter you wish to exclude from the number of pages, you need to do a calculation²:

- If you know the number of pages in the front matter (and it will not change), then you can use Page { Page } of { = { NumPages } - x }, where x is the number of pages in the front matter. For example:

`{= { NumPages } - 3 }`

(toggle field codes to see)

- If not, then you insert a bookmark on the last page of the document and use a PageRef field to reference the page number of that bookmark instead of the NumPages field.

Macros

The default behaviour of MergeDocx is to produce an output docx which contains no macros.

You can configure DocumentBuilder to retain the macros present in one of the source documents. To do this, you need to be using the BlockRange approach.

DocumentBuilder contains:

```
/**
 * With this setting, you can embed macros from one of the input documents, in the output docx.
 * Without it, macros will simply be ignored.
 * The macros come from the docm or dotm underlying the specified BlockRange.
 * The setting will be ignored if a docx or dotx underlies the specified BlockRange.
 * @param br
 */
public void setRetainMacros(BlockRange br)
```

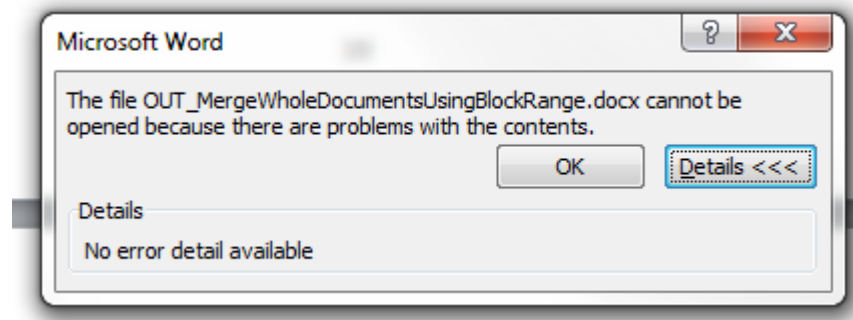
So you can do something like:

```
documentBuilder.setRetainMacros(blockRanges.get(2));
```

² <http://www.eggheadcafe.com/microsoft/Word-Page-Layout/35979216/total-page-number-minus-number-of-pages-in-front-matter.aspx>
http://wordribbon.tips.net/T010604_Field_Reference_to_Number_of_Prior_Pages.html

to keep the macros from docm/dotm underlying the 3rd BlockRange.

If MergeDocx finds macros in that block range, the resulting output document will be set to be of the same type (ie docm or dotm). *It is your responsibility*, when saving your output WordprocessingMLPackage, to save it with the correct filename extension. If a docm is saved with a docx extension, if you try to open it in Word 2010, you will an error similar to the following:



So you need to ensure you use the correct filename extension.

Interaction between ODD_PAGE and Page Number restart

With MergeDocx, you can use the settings described above to have each new document start on the right (recto) page, with numbering starting again from one:

```
block.setSectionBreakBefore(SectionBreakBefore.ODD_PAGE);  
block.setRestartPageNumbering(true);
```

Microsoft Word will not however, honour this combination, unless the docx is “tweaked” to make it do so.

There are two different ways MergeDocx can tweak the output docx in order to have Word behave as expected. You’ll need to experiment with both approaches; this is best done by physically printing the output from Word to your printer or to PDF. (You can print 4 pages per side to save paper, and still see what is going on.)

The first is:

```
documentBuilder.setSectionBreak_ODD_PAGE(  
    BEHAVIOUR_SectionBreak_ODD_PAGE.MIRROR_MARGINS);
```

This is the cleanest approach, and should be used where possible. For it to work, you need to ensure your first docx being merged has a document settings part (since the mirror margins setting is stored in that part, and MergeDocx gets that part from the first docx).

The second is:

```
documentBuilder.setSectionBreak_ODD_PAGE(  
    BEHAVIOUR_SectionBreak_ODD_PAGE.FIELD_IF_MOD);
```

If you use this approach, MergeDocx will insert an arcane field into your docx before appropriate sections (hit Shift F9 to see field codes):

{ IF { =MOD({ PAGE * Arabic },2)}= 0 " " "	
" }	Page Break
	Section Break (Odd Page)

The table below summarises the advantages and disadvantages of each approach:

<i>MIRROR_MARGINS</i>	+ doesn't introduce fields into the docx	- may not work if documents contain both portrait and landscape pages; see http://support.microsoft.com/kb/185528 - first docx must have a document settings part for this to work (you can add one with docx4j if it doesn't) - single setting per docx (though the other approach is the same in practice)
<i>FIELD_IF_MOD</i>	+ suited to a mixture of portrait and landscape pages + could in principle control each docx separately (contact Plutext if you need this) + can be adjusted to include "this page intentionally left blank"	- PDF output systems (other than Word) are less likely to support

Bullets & Numbering

When documents using the "same" numbering are merged, by default, the numbering will continue, not restart.

This is useful if you are merging chapters of a book, or sections of a contract, and you want the numbering to continue.

Sometimes however, you may want to force the numbering to restart. To do this, you instruct MergeDocx to add new lists, rather than re-using existing lists.

To do this, NumberingHandler to *ADD_NEW_LIST*:

```
BlockRange blockRange1 = ...
BlockRange blockRange2 = ...

source2.setNumberingHandler(NumberingHandler.ADD_NEW_LIST);
```

The default is *USE_EARLIER_IFF_SAME*. "same" means the formatting definition is the same (ie they look the same), and the list is based on the same abstract numbering definition identifier (nsid).

There is a third option, *USE_EARLIER*, which will use a list with the same nsid from an earlier BlockRange, irrespective of whether it looks the same. The numbering will continue, not restart. For example if the numbering of the list in the first BlockRange was decimal, and the second BlockRange contained a list with the same nsid but roman numbering, applying the *USE_EARLIER* to the second BlockRange would cause its numbering to be decimal (rather than roman).

EVENT MONITORING

Since merging documents can take some time (depending on the number and complexity of the documents), the possibility exists (new in **3.1.0**) of performing the merge in the background, and receiving notification when the job is complete.

See the MergeDocxProgress sample for an example of usage.

As per that example, you need to:

- create a message bus, and tell Docx4jEvent to use it
- define and register/suscribe a listener

This is done as follows:

```
// Creation of message bus
MBassador<Docx4jEvent> bus = new MBassador<Docx4jEvent>(
    BusConfiguration.Default());
// and registration of listeners
ListeningBean listener = new ListeningBean();
bus.subscribe(listener);
// tell Docx4jEvent to use your message bus for notifications
Docx4jEvent.setEventNotifier(bus);
```

The sample class contains an example ListeningBean. Note the @Handler annotation.

Docx4j's approach to event monitoring relies on the MBassador library; see further

<https://github.com/bennidi/mbassador>

For another example of monitoring events (docx load, save), please see

<https://github.com/plutext/docx4j/blob/master/src/samples/docx4j/org/docx4j/samples/EventMonitoringDemo.java>

ADVANCED TOPICS

The following topics document several features of the file format for interested readers. However, no special action should be necessary (beyond setting StyleHandler as appropriate) since MergeDocx should handle these quirks.

overrideTableStyleFontSizeAndJustification

In a document created in Word, the settings part, by default contains:

```
<w:compatSetting w:name="overrideTableStyleFontSizeAndJustification" .. w:val="1"/>
```

but this may vary by input document.

Where it is false, then anything in a table where font size 11/12 or jc left came from the Normal style was ignored (in favour of whatever the table style specified).

In the output docx, this is always set, so paragraph styles **do** override table styles.

Where that wasn't true in a particular input document, appropriate adjustments are made.

Document Defaults

The styles part of a docx contains an element called `w:docDefaults`. Example contents:

```

<w:docDefaults>
  <w:rPrDefault>
    <w:rPr>
      <w:rFonts w:asciiTheme="minorHAnsi" w:eastAsiaTheme="minorEastAsia"
w:hAnsiTheme="minorHAnsi" w:cstheme="minorBidi"/>
      <w:sz w:val="22"/>
      <w:szCs w:val="22"/>
      <w:lang w:val="en-US" w:eastAsia="ko-KR" w:bidi="ar-SA"/>
    </w:rPr>
  </w:rPrDefault>
  <w:pPrDefault>
    <w:pPr>
      <w:spacing w:after="200" w:line="276" w:lineRule="auto"/>
    </w:pPr>
  </w:pPrDefault>
</w:docDefaults>

```

These are the basic/root settings, on which the formatting/appearance is based. See further below for tips on seeing/manipulating `w:docDefaults`

When documents are merged, there can only be one `w:docDefaults` element.

If one or more blockrange have `StyleHandler.RENAME_RETAIN` (that is, you want to retain the existing look of each individual document), or incremental processing is being used, we merge the properties in doc defaults into the styles.

Editing Document Defaults

Microsoft Word provides ways to edit your document defaults, but no easy way to be sure what the settings are (since the Word interface conflates the default paragraph style (eg Normal) and DocDefaults/pPrDefault!).

To see the actual settings, we recommend looking at the raw XML. There are a few different ways to do this:

In Java

```

// Given WordprocessingMLPackage
org.docx4j.wml.Styles styles =
(org.docx4j.wml.Styles)wmlPkg.getMainDocumentPart().getStyleDefinitionsPart().getJaxbElement();
System.out.println(
    org.docx4j.XmlUtils.marshalToString(styles.getDocDefaults()));

```

or just:

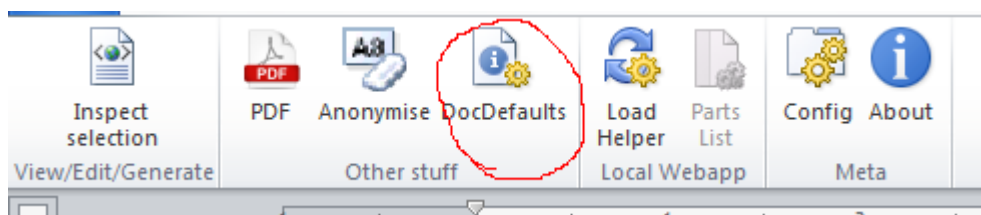
```

System.out.println(
    wmlPkg.getMainDocumentPart().getStyleDefinitionsPart().getXML() );

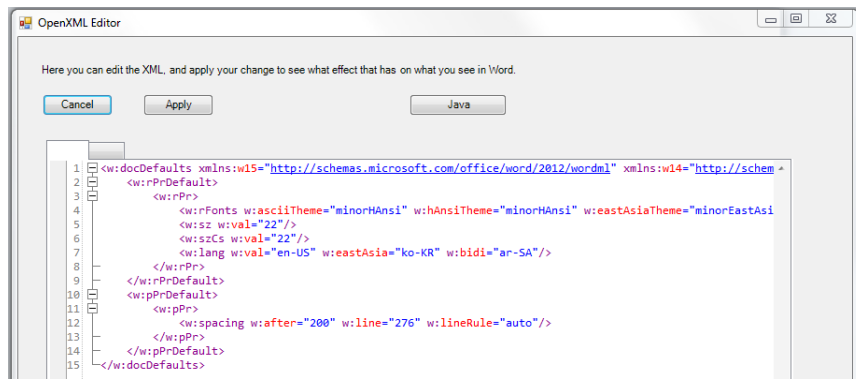
```

They'll be at the top.

or use the Docx4j Helper Word Addin (v3.3)



Clicking that, you'll see your `w:docDefaults` in an editor window:



If you edit the XML then click the apply button, the result will be a new docx containing your new settings.

or, unzip the docx, then open styles.xml

or use the webapp, to navigate to the styles part

or, if you have Visual Studio,

use the Open XML Package Editor for Visual Studio:

<https://visualstudiogallery.msdn.microsoft.com/450a00e3-5a7d-4776-be2c-8aa8cec2a75b>

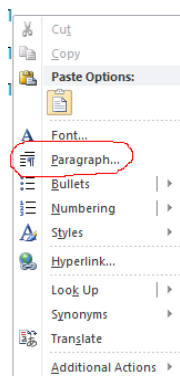
With that you can drag your docx onto Visual Studio, then navigate the tree to the styles part.

You can edit and save your changes.

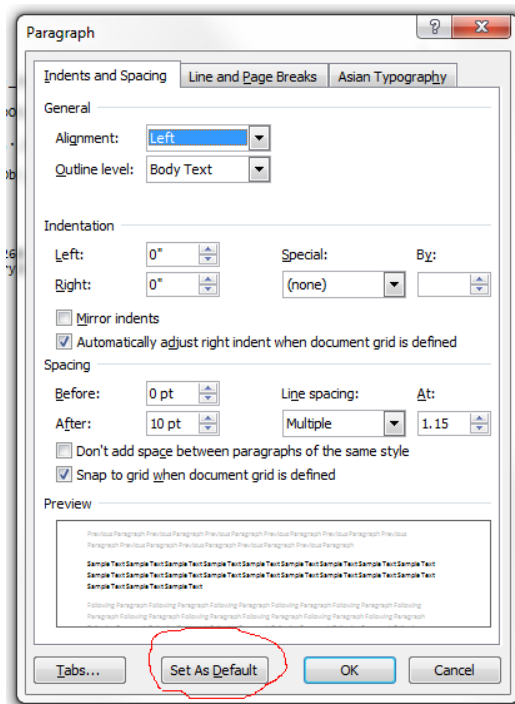
With some of the above approaches, you can edit your `w:docDefaults`.

Alternatively, you can do this in Word:

- To set paragraph level doc default properties, right click then choose “Paragraph” from the context menu.

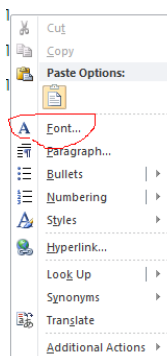


You should see:

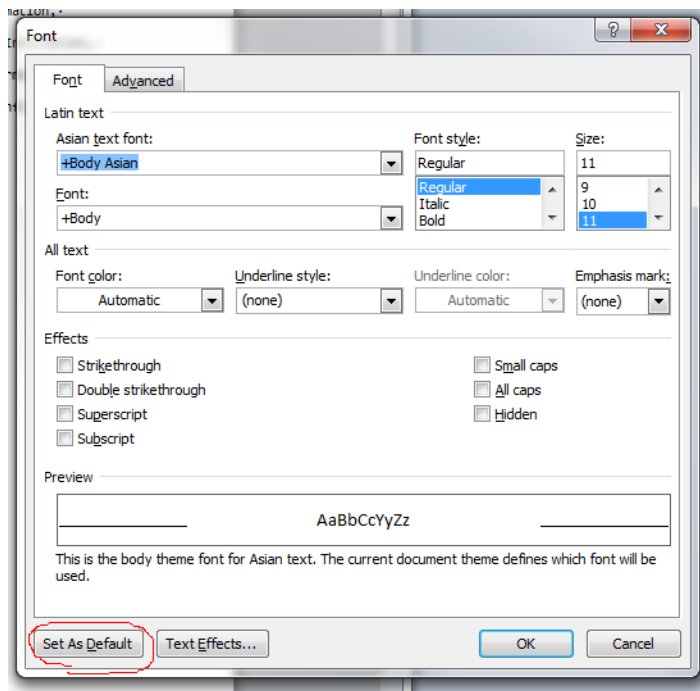


The key is the "**set as default**" button.

- To set run level doc default properties, right click then choose "Font" from the context menu.



Again, when you have things set as you wish, click the "*set as default*" button.



OLE Helper (for docx, pptx, xlsx)

This chapter explains how to use the docx4j OLE Helper tool

OLE Helper makes it easy to use docx4j to programmatically embed or link files as OLE objects in a docx, pptx, or xlsx.

File types which can be embedded/linked with this tool include:

- PDF
- Zip files
- Text files (plain text, CSV, XML etc)
- HTML and MHT files
- Open Document Format (LibreOffice/OpenOffice) files
- Binary Office documents (eg doc/ppt/vsd/xls files)
- Microsoft Project files
- Outlook message files
- Video files (.MOV, .WMV)

Without this OLE Helper, it can be a real challenge to convert your file into a suitably structured OLE object, which works across Office 2007, 2010 and 2013.

DOCX, PPTX, XLSX SUPPORT

You can use the OLE helper functionality to embed files in any of:

- docx
- pptx
- vsdx (Visio)
- xlsx

REQUIREMENTS/DEPENDENCIES

OLE Helper requires JPedal (to create an image from the first page of a PDF). This may be found in the lib dir.

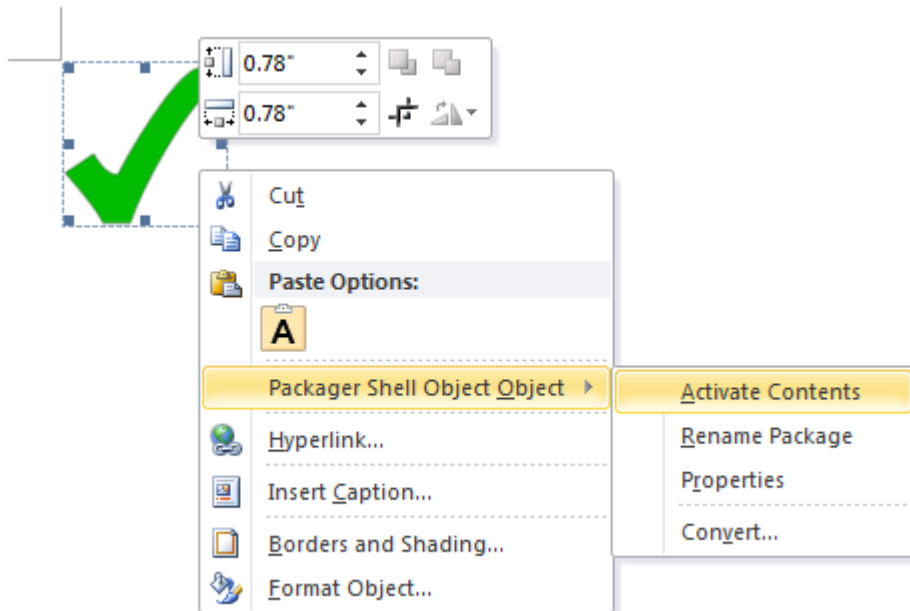
OLE CONCEPTS

OLE Linking and Embedding in Microsoft Office

A file can be linked from or embedded in an Office document (docx, pptx, xlsx).

Such a file will be represented by an icon or image on the document surface.

The user can right click on the object and “activate contents”:-



If the user has the necessary application installed, the file will then be opened in that application.

Double clicking is equivalent to selecting “Activate Contents”.

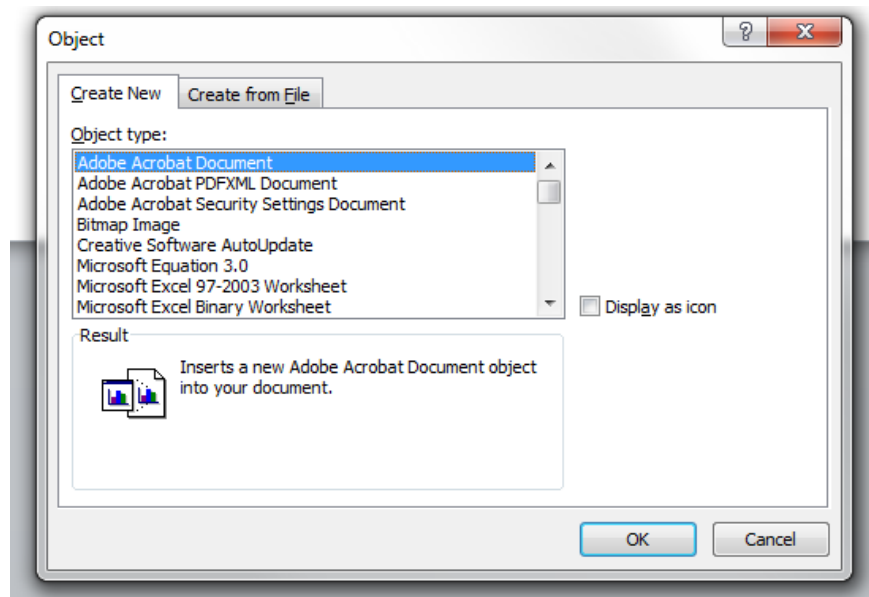
If the object is embedded, it is the object physically embedded in the Office document which is copied to a temp location and then opened.

If the object is linked, the object is opened from the location specified in the link. Obviously it needs to be available at that location for this to work.

If, after activation, you see the image change size, this is because Office 2010 automatically resizes the image if its specified size differs from its intrinsic size.

A document author can embed or link a document in this way, via the user interface of Word, Powerpoint or Excel (Insert > Object).

In Word, when you do this, you see the following dialog:



Excel and Powerpoint are similar. There are certain entries in the list which are always there, because Office knows what to do with them natively.

Others only appear there if you have suitable software installed. “Adobe Acrobat Document” is an example of this:- it only appears if you have at least Acrobat Reader installed.

Docx4j-OLE Helper allows you to programmatically link/embed objects, without any need for such things to be installed. The user opening the document will however need the relevant application should they wish to activate the object.

The OpenXML Specification

The Open XML specification states that the embedded object is represented by an “Embedded Object” part. The equivalent docx4j object is `org.docx4j.openpackaging.parts.WordprocessingML.OleObjectBinaryPart`. This object is used for docx, pptx and xlsx.

The spec says that for a docx, you can embed an object in the following parts:

- Main Document
- Comments
- Footnotes, Endnotes
- Header, Footer

For an XLSX, it can go in a Worksheet part.

In a pptx, it can go in:

- Slide
- Slide Layout
- Slide Master
- Handout Master
- Notes Slide
- Notes Master

Different XML is used, depending on whether it is being embedded in a docx, pptx, or xlsx.

Included below is sample XML for each of these contexts.

Docx

```
<w:r>
  <w:object>
    <v:shapetype id="_x0000_t75" coordsize="21600,21600" o:spt="75"
      o:preferrelative="t" path="m@4@5l@4@11@9@11@9@5xe"
      filled="f" stroked="f">
      <v:stroke joinstyle="miter"/>
    <v:formulas>
      <v:f eqn="if lineDrawn pixelLineWidth 0"/>
      <v:f eqn="sum @0 1 0"/>
      <v:f eqn="sum 0 0 @1"/>
      <v:f eqn="prod @2 1 2"/>
      <v:f eqn="prod @3 21600 pixelWidth"/>
      <v:f eqn="prod @3 21600 pixelHeight"/>
      <v:f eqn="sum @0 0 1"/>
      <v:f eqn="prod @6 1 2"/>
      <v:f eqn="prod @7 21600 pixelWidth"/>
    </v:formulas>
  </v:shapetype>
</w:object>
</w:r>
```

```

        <v:f eqn="sum @8 21600 0"/>
        <v:f eqn="prod @7 21600 pixelHeight"/>
        <v:f eqn="sum @10 21600 0"/>
    </v:formulas>
    <v:path o:extrusionok="f" gradientshapeok="t" o:connecttype="rect"/>
    <o:lock v:ext="edit" aspectratio="t"/>
</v:shapetype>
<v:shape id="_x0000_i1025" type="#_x0000_t75" o:ole="">
    <v:imagedata r:id="rId5" o:title=""/>
</v:shape>
<o:OLEObject Type="Embed" ProgID="AcroExch.Document.7"
    ShapeID="_x0000_i1025" DrawAspect="Icon"
    ObjectID="_1430550695" r:id="rId6"/>
</w:object>
</w:r>

```

Pptx

```

<a:graphic>
  <a:graphicData uri="http://schemas.openxmlformats.org/presentationml/2006/ole">
    <p:oleObj name="Acrobat Document" r:id="rId3"
      imgW="5667119" imgH="8019810"
      progId="AcroExch.Document.11">
      <p:embed/>
      <p:pic>
        <p:nvPicPr>
          <p:cNvPr id="0" name=""/>
          <p:cNvPicPr/>
          <p:nvPr/>
        </p:nvPicPr>
        <p:blipFill>
          <a:blip r:embed="rId4"/>
          <a:stretch>
            <a:fillRect/>
          </a:stretch>
        </p:blipFill>
        <p:spPr>
          <a:xfrm>
            <a:off x="4181475" y="719138"/>
            <a:ext cx="3829050" cy="5418137"/>
          </a:xfrm>
          <a:prstGeom prst="rect">
            <a:avLst/>
          </a:prstGeom>
        </p:spPr>
      </p:pic>
    </p:oleObj>
  </a:graphicData>
</a:graphic>

```

Xlsx

```

<legacyDrawing r:id="rId1"/>
<oleObjects>
  <oleObject r:id="rId2" shapeId="1025" progId="AcroExch.Document.11" />
</oleObjects>

```

Each of the above has in common:

- A relID pointing to the Embedded Object part
- A relID pointing to an image
- A progID

For pptx, you also specify the location and size of the image

The docx4j-OLE Helper helps you to create the above structures, but its main contribution is to put your file in the correct format

USAGE

Overview

A high level API is provided, which allows you to:

- Link, or
- Embed

in a docx, pptx, or xlsx:

- a PDF, or
- an ODF (Open Document Format file, as used in LibreOffice/OpenOffice)
- other object.

PDF and ODF handling/behaviour is a bit different.

In each case, an image is required. Office default behaviour is to use a file icon, except in the case of PDF, where the first page is shown. The API gives you similar options:

- allow it to generate and use a file icon (for PDF the methods have suffix **UsingIcon**),
- provide a specific image you wish to use (which goes beyond what the Office user interface offers) (for PDF the methods have suffix **UsingImage**), or
- (in the case of PDF only) generate the image from the first page of the object

The API is contained in the following helper classes:

	OLE object		
	PDF	ODF	Other
Docx	PdfOleHelperDocx	OdfOleHelperDocx	OleHelperDocx
Pptx	PdfOleHelperPptx	OdfOleHelperPptx	OleHelperPptx
Xlsx	PdfOleHelperXlsx	OdfOleHelperXlsx	OleHelperXlsx

Note the following parameters:

Caption (Label)	A text string which may appear on the surface of the document. Typically, the file name.
File path	Typically, the complete path of the original file location
Command	For embed , this is typically the complete path of the file (including file name). Only the file extension matters, since that may be used to determine what program opens the object (determined on the client computer by its Control Panel\Programs\Default Programs\Set Associations). The dir path is typically replaced by the actual location of the user's temp dir. For link , use the complete path of the original file location

For objects other than a PDF, you need to specify the embedding type. `enum EmbeddingType` includes:

```
DOC,  
PPT,  
VSD,  
XLS,  
DOCX,  
PPTX,  
VSDX,  
XLSX,  
  
MPP,  
  
MMAP(mindmap),  
  
// One Note  
ONE, ONE_PKG(Notebook),  
  
MSG,  
PDF,  
  
HTML,  
MHT,  
  
TXT,  
CSV,  
XML,  
  
ZIP,  
  
MOV,  
WAV,  
WMV,  
  
ODP,  
ODS,  
ODT,
```

Linking/embedding in a docx

To link/embed in a docx, use `OleHelperDocx` (or `PdfOleHelperDocx` if you are embedding a PDF).

The `OleHelperDocx` API returns a run (w:r) object containing the OLE object, suitable for embedding in a paragraph (w:p) anywhere in the main document part.

If you are inserting a PDF:

- The plain embed/link methods in `PdfOleHelperDocx` will generate an image from the PDF
- You can use the methods with suffix **UsingIcon** to generate a file icon image, with the caption/label you provide
- You can use the methods with suffix **UsingImage** to supply some arbitrary image of your own. Since you have to supply a PDF in the embedding case (as opposed to linking), that PDF can be used to generate an image if you supply null.

If you are inserting some other type of object:

- If you don't supply an image, a file icon image will be generated, with the caption/label you provide

EmbeddingType	Sample name	Notes
PDF	Docx_ole_PDF_icon.java	Embed using generated icon
PDF	Docx_ole_PDF.java	Embed creating image from first page of PDF
DOC, PPT, XLS	Docx_ole_Doc.java	Demo using custom image
VSD	Docx_ole_vsd.java	
DOCX, PPTX, XLSX	Docx_ole_Xlsx.java	
VSDX	Docx_ole_vsdX.java	
MPP	Docx_ole_MicrosoftProject.java	
MMAP	Docx_ole_MindMap.java	
MSG	Docx_ole_OutlookMsg.java	
ONE	Docx_ole_OneNote.java	
ONE_PKG	Docx_ole_OneNotePkg.java	
ODF	Docx_ole_ODF.java	
HTML	Docx_ole_HTML.java	
MHT	Docx_ole_MHT_WebArchive.java	
CSV	Docx_ole_csv.java	Demo using custom image
TXT	Docx_ole_txt.java	
ZIP	Docx_ole_zip.java	
WAV	Docx_ole_wav.java	
MOV, WMV	Docx_ole_wmv.java	
Package	Docx_ole_Binary.java	For arbitrary file type

Some of the signatures include a **style** parameter, which you can use to specify image **size** (eg "width:446pt;height:630pt"). You don't really need to worry about this, unless you supply your own image.

If you do supply your own image, the following considerations apply. Word 2010 will initially display the image using the style you specify. However, after the user activates the OLE object, the image will be resized to its intrinsic size (if that is different).

So if you want to avoid Word resizing the image:

- you can supply a style parameter which specifies the correct size, or
- you can allow docx4j to work it out (supply a null style)

Best practise is to alter the intrinsic size of your image (resizing with your preferred library) before invoking PdfOleHelperDocx.

Docx4j can automatically work out what style to use, by examining the image. But that will cause a temp file to be created.

To avoid that, you need to specify that size via the style parameter, and the mime type. For example, a 96 pixels per inch image (ThumbnailViaJPedal creates an image 595px height by 841px wide at 96 pixels/inch) can be converted to points by multiplying by 72/96.

If you don't care whether the image is resized, you can:

- leave the style null, in which case unless docx4j works out a style for you, Word displays by default at 0.7" x 0.7", or
- you can supply a custom size, which Word will use

Rule of thumb if you supply your own image is to: supply style and mime type which are both null, or both non-null. Image introspection will occur if mime type is null.

Linking/embedding in a pptx

To link/embed in a pptx, use OleHelperPptx (or PdfOleHelperPptx if you are embedding a PDF, or OdfOleHelperPptx if you are embedding an ODF).

You'll have to provide the coordinates at which the image is to be inserted, by passing the top, left position (each an int, in points).

When using OleHelperPptx, a caption of the form "aaa (file.txt)" should be avoided, since upon activation, Powerpoint (2010 and 2013 at least), convert the caption to the form "caption (caption.txt)".

When embedding a PDF, you can choose to show the first page of the PDF, or to use an icon.

An ODF, can on activation, be edited in-place.

Other file types will typically show as an icon after activation.

Linking/embedding in a xlsx

To link/embed in xlsx, use OleHelperXlsx (or PdfOleHelperXlsx if you are embedding a PDF).

Excel 2010 will scale the image to match anchorLocation.

You have to provide an array of integers, specifying the anchor location for the object. This corresponds to the [VML anchor element](#). The values are: LeftColumn, LeftOffset, TopRow, TopOffset, RightColumn, RightOffset, BottomRow, BottomOffset.

Value	Description
LeftColumn	The left anchor column of the object (left-most column is 0). [Example: An object whose left anchor was off of the third column would have a LeftColumn value of 2. end example]
LeftOffset	The offset of the object's left edge from the left edge of the left anchor column. This value is measured in pixels.
TopRow	The top anchor row of the object (top-most column is 0). [Example: An object whose top anchor was off of the fifth row would have a TopRow value of 4. end example]
TopOffset	The offset of the object's top edge from the top edge of the top anchor row. This value is measured in pixels.
RightColumn	The right anchor column of the object (left-most column is 0). [Example: An object whose right anchor was off of the tenth column would have a RightColumn value of 9. end example]
RightOffset	The offset of the object's right edge from the left edge of the right anchor column. This value is measured in pixels.
BottomRow	The bottom anchor row of the object (top-most column is 0).

Value	Description
	[Example: An object whose bottom anchor was off of the tenth row would have a BottomRow value of 9. end example]
BottomOffset	The offset of the object's bottom edge from the bottom edge of the bottom anchor row. This value is measured in pixels.

[Example: The left side of the object is 15 pixels to the right of the left edge of the second column. The top edge is 2 pixels below the upper edge of the first row. The right side is 15 pixels to the right of the left edge of the fourth column. The bottom edge is 16 pixels below the top of the fourth row.

```
int[] anchorLocation = {1, 15, 0, 2, 3, 15, 3, 16}
```

When using a file type icon, this looks best if it is around 3 rows high (at standard row height). The width (number of columns) depends on the length of the caption string. The following code snippet accommodates that:

```
int cols = (int)Math.max(1, Math.round( caption.length()/5)); // accommodate long caption
int leftCol=4;
int rightCol=leftCol+cols;
int topRow=4;
int bottomRow=topRow+3; // fixed at 3 rows high, good for a file icon

int[] anchorLocation = {leftCol, 0, topRow, 0, rightCol, 0, bottomRow, 0};
```

SAMPLE CODE

Sample code is included to demonstrate various cases.

Note that the only difference between linking and embedding, is whether the content is included. Because of this, each sample can be readily used for either linking or embedding.

EmbeddingType	docx	pptx	xlsx
PDF	Docx_ole_PDF_icon.java	Pptx_ole_PDF_icon.java	Xlsx_ole_PDF_icon.java
PDF	Docx_ole_PDF.java	Pptx_ole_PDF.java	Xlsx_ole_PDF.java
DOC, PPT, XLS VSD	Docx_ole_Doc.java Docx_ole_vsd.java		
DOCX, PPTX, XLSX VSDX	Docx_ole_Xlsx.java Docx_ole_vsdX.java	Pptx_ole_Xlsx.java Pptx_ole_vsdX.java	Xlsx_ole_docx.java
MPP	Docx_ole_MicrosoftProject.java		
MMAP	Docx_ole_MindMap.java		
MSG	Docx_ole_OutlookMsg.java		
ONE ONE_PKG	Docx_ole_OneNote.java Docx_ole_OneNotePkg.java		
ODF	Docx_ole_ODF.java	Pptx_ole_ODF.java	Xlsx_ole_ODF.java
HTML	Docx_ole_HTML.java		
MHT	Docx_ole_MHT_WebArchive.java		
CSV	Docx_ole_csv.java		
TXT	Docx_ole_txt.java	Pptx_ole_txt.java	Xlsx_ole_txt.java
ZIP	Docx_ole_zip.java		
WAV	Docx_ole_wav.java		

MOV, WMV	Docx_ole_wmv.java	Pptx_ole_wmv.java	Xlsx_ole_wmv.java
Package	Docx_ole_Binary.java	Docx_ole_Binary.java	Docx_ole_Binary.java

NEW FILE TYPES

Supporting a new file type is essentially a matter of creating a .cfb template file for it.

If you'd like a new file type to be supported, please email a sample docx in which you have embedded (using Word) a file of that type, to support@plutext.com

KNOWN ISSUES/LIMITATIONS

Word 2007

Icons shrink after activation.

Powerpoint 2010 x64

Activating a PDF which is shown as an icon should work. However, if the first page of the PDF is displayed, after activation, it may be replaced in the slide with a blank image.

Office 2010 support for ODF

Office 2010 only supports ODF 1.1 (Office 2013 can read ODF 1.2 files).

Bear in mind that recent versions of OpenOffice and LibreOffice (eg OpenOffice 4.0.14 and LibreOffice 4.1.4) create ODF 1.2 files. Before embedding an ODF file, you are advised to manually open it in Office 2010 to verify that that works.

Office 2011 and 2016 for Mac OSX

Support for OLE activation in Office on OSX is severely limited. It only knows how to open Office document types (docx/pptx/xlsx, and binary doc/ppt/xls), and can't readily activate PDF, text etc.

Office Online

In Word Online, you won't see an OLE object (these are silently dropped by Word Online). (Tested with an embedded PDF in December 2019). Expect similar behaviour in Powerpoint and Excel Online.

Digital Signatures (for docx, pptx, xlsx)

INTRODUCTION

This chapter explains how to use the docx4j dig sig tool:

Dig Sig makes it easy to use docx4j to work with digital signatures:

- programmatically sign docx, pptx, or xlsx files
 - with 1 or more signatures
 - with or without the signature being “visible” (docx only)
- validate signatures

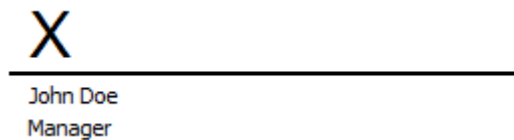
The implementation follows the Microsoft spec, so the signatures work as expected in Microsoft Office.

SIGNATURES IN MICROSOFT OFFICE

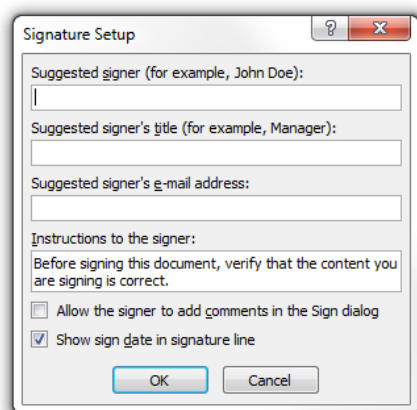
A signature can be visible (in Excel or Word) or invisible.

In either case, the signature also hashes the document (more on this below), so Office can detect whether its been “tampered” with (ie edited) after signing. This is its primary purpose: to detect whether signed data has been altered

In Word, a “visible” signature looks like this:

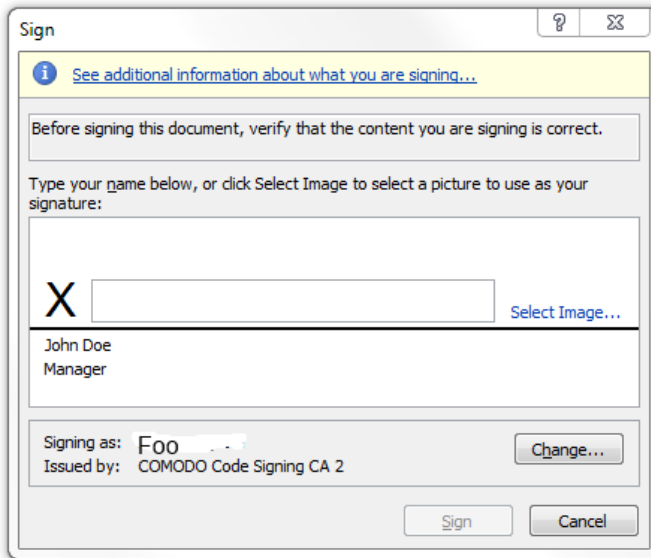


This is inserted using the Signature Line dialog (Insert > Singature Line):

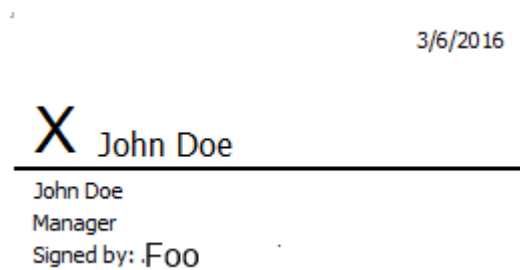


A user can click on the Signature Line to sign it.

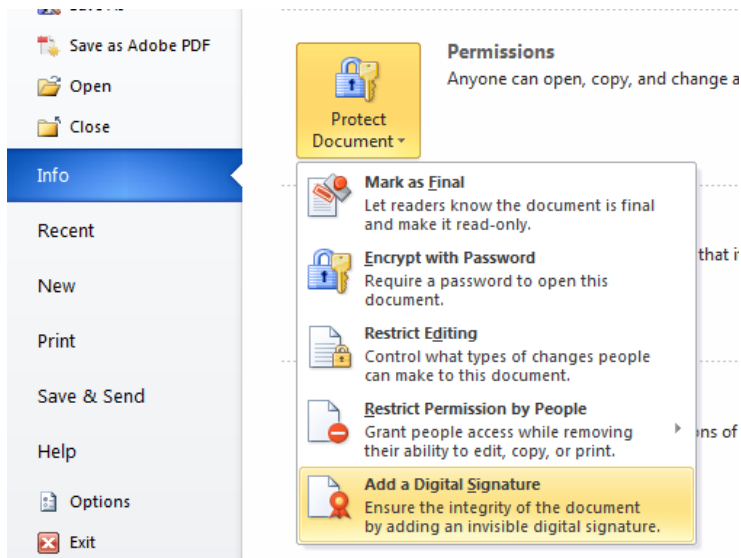
First they'll see a dialog:



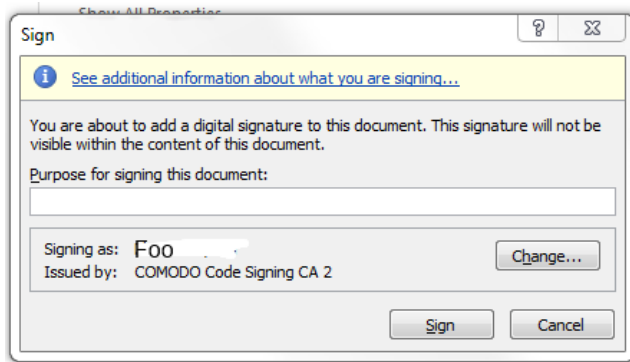
When someone signs it, their signature appears there:



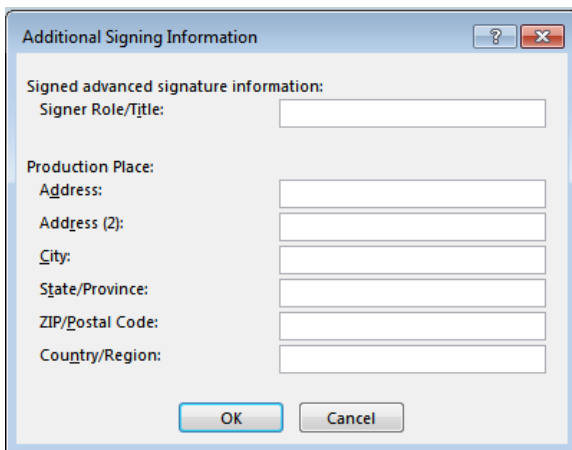
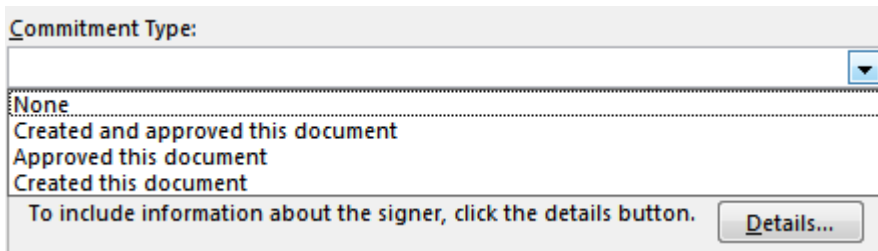
Signatures can also be “invisible”. In this case, the document is hashed, but there is no signature on the document surface. You add an invisible digital signature via File>Info>Protect Document:



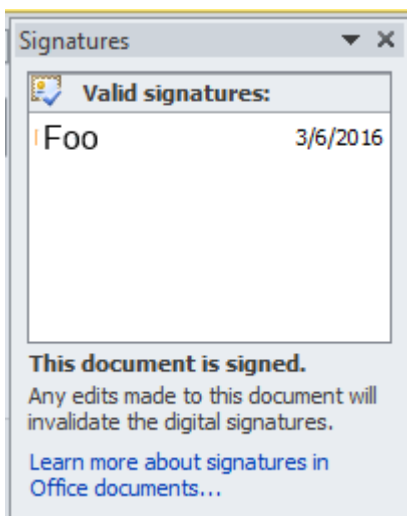
The sign dialog is similar:



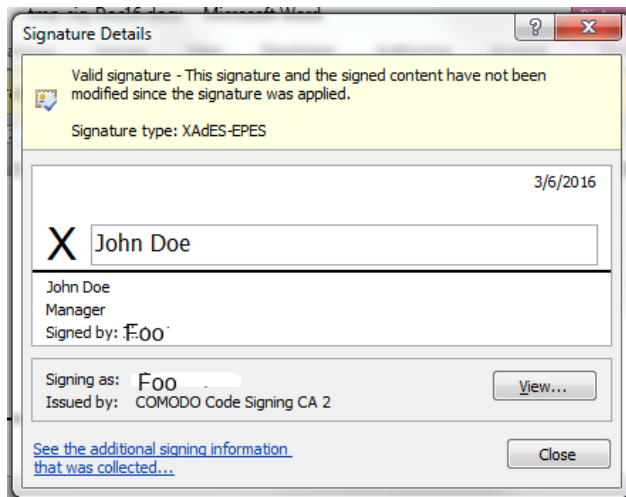
In Word 2013/2016, this dialog has some additional fields:



Either way, after a document has been signed, the Signature panel gives you visual confirmation that the signatures are valid/intact (that is, that the signed parts have not been altered):

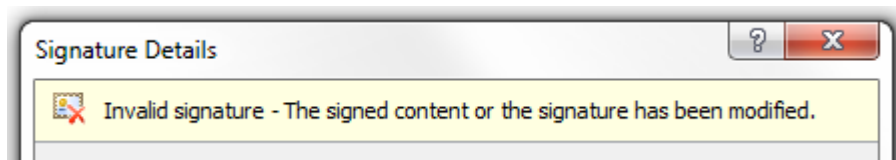
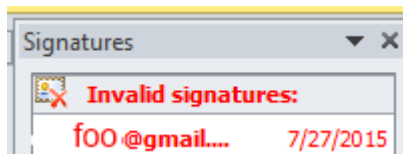


You can click to get the details:

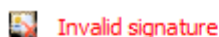


(The information given differs a little depending on whether the signature is visible or not)

You are alerted if the signature is no longer valid:



In the case of a visual signature, you will also see red text superimposed on it:



X John Doe

John Doe
Manager
Signed by: John Doe, Manager

SIGNATURE SUPPORT IN OFFICE

Signatures are supported in Office 2007, 2010 and 2013, however, Powerpoint does not support visible signatures.

Signatures are not supported in Office for Mac (either 2011 or 2016).

THE SPECIFICATION

Signing is part of the Open Packaging Conventions spec.

Docx4j Enterprise can sign documents following that spec; it can also validate signatures.

As noted in the spec:

Digital signatures do not protect data from being changed. However, consumers can detect whether signed data has been altered and notify the end-user, restrict the display of altered content, or take other actions.

It isn't necessary to understand the full details in order to use Docx4j's digital signature functionality. However, it is useful to understand the role of XmlSignaturePart:

- when you sign a document, you create an XmlSignaturePart
- there is an XmlSignaturePart for each signature
- the XmlSignaturePart is used for validation

An XmlSignaturePart looks something like:

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#"
xmlns:dssi="http://schemas.microsoft.com/office/2006/digsig"
xmlns:mdssi="http://schemas.openxmlformats.org/package/2006/digital-signature"
xmlns:xd="http://uri.etsi.org/01903/v1.3.2#" Id="idPackageSignature">
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xm1-c14n-
20010315"></CanonicalizationMethod>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"></SignatureMethod>
    <Reference Type="http://www.w3.org/2000/09/xmldsig#Object" URI="#idPackageObject">
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></DigestMethod>
      <DigestValue>z7n7di9KSx4Vqt0fph4aLW2f2v8=</DigestValue>
    </Reference>
    <Reference Type="http://www.w3.org/2000/09/xmldsig#Object" URI="#idOfficeObject">
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></DigestMethod>
      <DigestValue>Kj6v0MJbJZSjEvWCJTqJ9Ka1wis=</DigestValue>
    </Reference>
    <Reference Type="http://uri.etsi.org/01903#SignedProperties" URI="#idSignedProperties">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/TR/2001/REC-xm1-c14n-20010315"></Transform>
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></DigestMethod>
      <DigestValue>ErjZGNguSuMk9yR5GarA7POLx4Y=</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue Id="idPackageSignature-signature-
value">ejB0ZSQY7fK4GcsNH2tfm2TlCWG+..qj003wvp48B0NaBD9GtG3P8jy2GbEsVM23ZaRaBRA==</SignatureValue>

  <KeyInfo> <!-- KeyInfoSignatureFacet -->
    <X509Data>

<X509Certificate>MIIFGCCBACgAwIBAgIQf9Hy8UJoFT4iDZtsk+..JhxM8k4JSm1Xwk980q87tkZ</X509Certificate>
    </X509Data>
  </KeyInfo>
  <Object Id="idPackageObject">
    <Manifest>
      <Reference URI="/word/document.xml?ContentType=application/vnd.openxmlformats-
officedocument.wordprocessingml.document.main+xml">
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></DigestMethod>
        <DigestValue>w8hL8o7msdLrxRjyoP0+OXvf0x8=</DigestValue>
      </Reference>
      :
    </Manifest>
    <SignatureProperties Id="id-signature-time-Wed Mar 02 11:11:48 EST 2016">
      <SignatureProperty Id="idSignatureTime" Target="#idPackageSignature">
        <mdssi:SignatureTime>
          <mdssi:Format>YYYY-MM-DDThh:mm:ssTZD</mdssi:Format>
          <mdssi:Value>2016-03-02T00:11:48Z</mdssi:Value>
        </mdssi:SignatureTime>
      </SignatureProperty>
    </SignatureProperties>
  </Object>
</Signature>
```

```

        </SignatureProperty>
    </SignatureProperties>
</Object>
<Object Id="idOfficeObject">
    <SignatureProperties>
        <SignatureProperty Id="idOfficeV1Details" Target="#idPackageSignature">
            <dssi:SignatureInfoV1>
                :
                <dssi:SignatureProviderDetails>0</dssi:SignatureProviderDetails>
                <dssi:SignatureType>1</dssi:SignatureType>
            </dssi:SignatureInfoV1>
        </SignatureProperty>
    </SignatureProperties>
</Object>

<Object>
    <xd:QualifyingProperties Target="#idPackageSignature">

        <xd:SignedProperties Id="idSignedProperties">
            <xd:SignedSignatureProperties>
                <xd:SigningTime>2016-02-02T11:11:48Z</xd:SigningTime>
                <xd:SigningCertificate>
                    <xd:Cert>
                        <xd:CertDigest>
                            <DigestMethod Algorithm="http://www.w3.org/2000/09/xmlsig#sha1"></DigestMethod>
                            <DigestValue>zk1lW0k+1aUEe7B+bZzg084hQy8=</DigestValue>
                        </xd:CertDigest>
                        <xd:IssuerSerial>
                            <X509IssuerName>[issuer]</X509IssuerName>
                            <X509SerialNumber>16990207296591082732488088650</X509SerialNumber>
                        </xd:IssuerSerial>
                    </xd:Cert>
                </xd:SigningCertificate>
                <xd:SignaturePolicyIdentifier>
                    <xd:SignaturePolicyImplied/>
                </xd:SignaturePolicyIdentifier>
                <xd:SignatureProductionPlace>
                    <xd:City></xd:City>
                    <xd:StateOrProvince></xd:StateOrProvince>
                    <xd:PostalCode></xd:PostalCode>
                    <xd:CountryName></xd:CountryName>
                </xd:SignatureProductionPlace>
                <xd:SignerRole>
                    <xd:ClaimedRoles>
                        <xd:ClaimedRole> </xd:ClaimedRole>
                    </xd:ClaimedRoles>
                </xd:SignerRole>
            </xd:SignedSignatureProperties>
            <xd:SignedDataObjectProperties>
                <xd:CommitmentTypeIndication>
                    <xd:CommitmentTypeId>
                        <xd:Identifier>http://uri.etsi.org/01903/v1.2.2#ProofOfOrigin</xd:Identifier>
                        <xd:Description>Created and approved this document</xd:Description>
                    </xd:CommitmentTypeId>
                    <xd:AllSignedDataObjects/>
                </xd:CommitmentTypeIndication>
            </xd:SignedDataObjectProperties>
        </xd:SignedProperties>

        <xd:UnsignedProperties>
            <xd:UnsignedSignatureProperties>
                <!-- Office 2013 XAdES-X-L, not implemented in Docx4j 3.3.0 -->
                <xd:CertificateValues>
                    <xd:EncapsulatedX509Certificate>MIIE5 .. y6vFUG=</xd:EncapsulatedX509Certificate>
                    <xd:EncapsulatedX509Certificate>MIIEZj ..s0AH8g=</xd:EncapsulatedX509Certificate>
                </xd:CertificateValues>
            </xd:UnsignedSignatureProperties>
        </xd:UnsignedProperties>

    </xd:QualifyingProperties>
</Object>
</Signature>

```

SPECIFICATION

The Open Packaging Conventions spec describes how the package digital signature framework applies the W3C Recommendation “XML-Signature Syntax and Processing” with certain modifications.

The W3C Recommendation <https://www.w3.org/TR/xmlsig-core/> explains how to generate a signature, and validate it. It is not necessary to understand the information in this section in any detail, in order to use Docx4j to sign or validate.

The following is quoted from the spec:

3.1 Core Generation

The REQUIRED steps include the generation of Reference elements and the SignatureValue over SignedInfo.

3.1.1 Reference Generation

For each data object being signed:

1. Apply the Transforms, as determined by the application, to the data object.
2. Calculate the digest value over the resulting data object.
3. Create a Reference element, including the (optional) identification of the data object, any (optional) transform elements, the digest algorithm and the DigestValue. (Note, it is the canonical form of these references that are signed in 3.1.2 and validated in 3.2.1 .)

3.1.2 Signature Generation

1. Create SignedInfo element with SignatureMethod, CanonicalizationMethod and Reference(s).
2. Canonicalize and then calculate the SignatureValue over SignedInfo based on algorithms specified in SignedInfo.
3. Construct the Signature element that includes SignedInfo, Object(s) (if desired, encoding may be different than that used for signing), KeyInfo (if required), and SignatureValue.

3.2 Core Validation

The REQUIRED steps of [core validation](#) include (1) [reference validation](#), the verification of the digest contained in each Reference in SignedInfo, and (2) the cryptographic [signature validation](#) of the signature calculated over SignedInfo.

3.2.1 Reference Validation

1. Canonicalize the SignedInfo element based on the CanonicalizationMethod in SignedInfo.
2. For each Reference in SignedInfo:
 1. Obtain the data object to be digested. (For example, the signature application may dereference the URI and execute Transforms provided by the signer in theReference element, or it may obtain the content through other means such as a local cache.)
 2. Digest the resulting data object using the DigestMethod specified in its Reference specification.
 3. Compare the generated digest value against DigestValue in the SignedInfo Reference; if there is any mismatch, validation fails.

3.2.2 Signature Validation

1. Obtain the keying information from [KeyInfo](#) or from an external source.

2. Obtain the canonical form of the SignatureMethod using the CanonicalizationMethod and use the result (and previously obtained KeyInfo) to confirm the SignatureValue over the SignedInfo element.

Note that the Open Packaging Conventions modify/elaborate on these steps slightly.

In addition, the Signature panel in Office also checks **certificate** validity (ie in addition to reference and signature validation).

XAdES

XAdES v1.4.1³ extends XMLDSIG into the domain of non-repudiation by defining XML formats for advanced electronic signatures that remain valid over long periods and are compliant with EU-DIR-ESIG. XAdES incorporates additional useful information, including evidence as to validity even if the signer or verifying party later attempts to deny (repudiates) the validity of the signature.

XAdES includes the following seven forms:

XAdES-BES (Base)	<i>provides basic authentication and integrity protection:</i> and satisfies the legal requirements for advanced electronic signatures, but does not provide non-repudiation of its existence.	Supported
XAdES-EPES	as above, but it has a SignaturePolicyIdentifier element. In Office, the SignaturePolicyIdentifier element only has the default element (SignaturePolicyImplied).	Default
XAdES-T	<i>with Time-Stamp:</i> Includes time-stamp to provide protection against repudiation	Supported
XAdES-C	<i>with complete validation data:</i> Includes references to the set of data supporting the validation of the electronic signature (i.e. the references to the certification path and its associated revocation status information). This form is useful for those situations where such information is archived by an external source, like a trusted service provider.	Not currently supported
XAdES-X	<i>with eXtended validation data:</i> Includes time-stamp on the references to the validation data or on the ds:Signature element and the aforementioned validation data. This time-stamp counters the risk that any keys used in the certificate chain or in the revocation status information may be compromised.	Not currently supported
XAdES-X-L	<i>with eXtended validation data incorporated for the long term:</i> Includes the validation data for those situations where the validation data are not stored elsewhere for the long term. CertificateValues: A verifier will have to prove that the certification path was valid, at the time of the validation of the signature, up to a trust point according to the naming constraints and the certificate policy constraints from the "Signature Validation Policy". It will be necessary to capture all the certificates from the	Could support (see CertificateTrustChecker further below) (Office 2013 & 2016 write

³ http://uri.etsi.org/01903/v1.4.1/ts_101903v010401p.pdf

certification path, starting with those from the signer and ending up with those of the certificate from one trusted root of the "Signature validation policy".

RevocationValues: When dealing with long term electronic signatures, all the revocation data used in the verification of such signatures must be stored and conveniently time-stamped

XAdES-A	<i>with archiving validation data:</i> It includes additional time-stamps for archiving signatures in a way that they are protected if the cryptographic data become weak.	(Not supported by Office)
---------	--	---------------------------

XAdES- EPES is the default format for Office 2010 signatures.

Docx4j follows this. The property `com.plutext.dsig.XAdES.Level` supports the following values:

- 0 - XAdES Off (Create XML-DSig signatures)
- 1 - Create XAdES-EPES signatures (default)
- 2 - Create XAdES-T signatures

Regarding XAdES-T, quoting https://blogs.msdn.microsoft.com/david_leblanc/2010/05/30/office-2010-digital-signatures-and-xades/

The next level, XAdES-T is where you really get a lot of benefit. A serious problem with XML-DSig signatures is that they're not good for the long term. You are typically given a certificate that's good for 1-2 years, you sign something, and then open it 2 years later, and it's invalid. We can't replace pen and paper this way – real signatures have to be good for many years. The solution is a `SignatureTimeStamp` element. This uses RFC 3161 to record a timestamp of the signature value. Assuming that we can trust the timestamp server, we now have external proof of the signing time. This means that a future expiration doesn't apply, and a revocation may not apply.

To create an XAdES-T signature, you need to do 2 things:

1. Set docx4j property `com.plutext.dsig.XAdES.Level=2` in docx4j.properties
2. Set your time stamp server:

```
signatureHelper.getSignatureConfig().setTspUrl(tspUrl);
```

and any other properties (username, proxy etc).

USAGE: INVISIBLE SIGNING

In this scenario you sign the package with one or more signatures (each creating an `XmlSignaturePart`), but the signatures are not visible on the document surface.

Basic usage (signature is not visible):

```
SignatureHelper signing= new SignatureHelper(pkg);
signing.configureSignature(fis, password);
signing.sign();
// then save your pkg
```

`signing.configureSignature` is used to provide the key to be used for signing the package. Several method signatures are available:

```
/**
 * Provide details of a PKCS12 key, in preparation for signing this package.
 */
public SignatureDetail configureSignature(InputStream PKCS12stream, char[] password) throws
Docx4JException {

    KeyStore keystore = KeyStore.getInstance("PKCS12"); // a new keystore
    keystore.load(PKCS12stream, password); // now containing one entry

    :

/**
 * Provide details of a key to be used for signing this package
 *
 * @param pkEntry
 */
public SignatureDetail configureSignature(KeyStore.PrivateKeyEntry pkEntry)
```

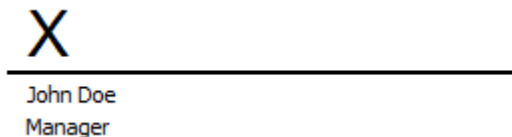
To use this latter method, load a keystore of your choice, then invoke its `getEntry` method. For more details, see <https://docs.oracle.com/javase/7/docs/api/java/security/KeyStore.html>

USAGE: VISIBLE SIGNATURE

A visible signature is a “signature line” on the document surface, coupled with an `XmlSignaturePart`.

The two are tied together via a reference from the `XmlSignaturePart` to the signature line’s ID.

Before signing, the signature line will look something like:



John Doe
Manager

The underlying XML:

```
<w:r>
  <w:pict>
    <v:shapetype id="_x0000_t75" coordsize="21600,21600" ...>
      :
    </v:shapetype>
    <v:shape id="_x0000_i1025" type="#_x0000_t75" alt="Microsoft Office
Signature Line..." style="width:192pt;height:96pt">
      <v:imagedata r:id="rId5" o:title=""/>
      <o:lock v:ext="edit" ungrouping="t" rotation="t" cropping="t"
vertices="t" text="t" grouping="t"/>
      <o:signatureline v:ext="edit" id="{12846971-C688-4F86-B09E-338A31F4D41B}"
o:suggestedsigner="John Doe"
o:suggestedsigner2="Manager" issignatureline="t"/>
    </v:shape>
  </w:pict>
</w:r>
```

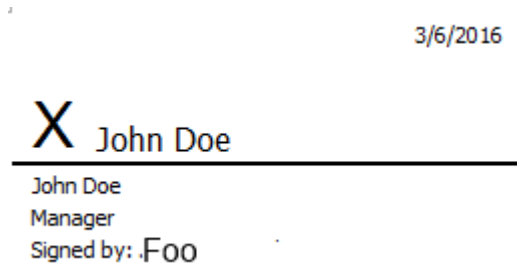
Note:

- the `o:signatureline` element
- the `rel id` in the `v:imagedata` element, which is to an EMF image

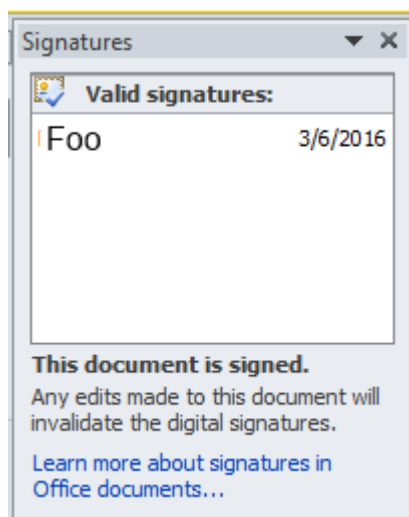
You can sign the document (ie add the XmlSignaturePart) at the same time as inserting the signature line, or you can do it later (via docx4j or using Word).

If the signature line is already present in the docx, you can use docx4j to sign the document (ie add the XmlSignaturePart).

After signing, the signature line will look something like:



and the signature panel should say:



Visible signatures are currently only supported for Docx files. (Excel 2010 has signature line support, but Docx4j does not currently have a high level API for creating an Excel signature line)

Inserting a signature line

```
SignatureHelper helper= new SignatureHelper(pkg);

CTSignatureLine sigLine = helper.createCTSignatureLine("John Doe", true);
P p = helper.createDocxSignatureLineP(sigLine, imageRelId, null, true);
pkg.getMainDocumentPart().addObject(p);
```

When you insert a signature line, Word stores an EMF image of the signature block. Note that Docx4j does not currently support generating a custom signature block EMF image for a user to use to later sign. However, this does not matter if signing is done later via the Office user interface, since Office generates that image itself on-the-fly. For this reason, the sample code just uses a tiny 2 pixel image.

CTSignatureLine represents `o:signatureline:`

```
<o:signatureline v:ext="edit" id="{12846971-C688-4F86-B09E-338A31F4D41B}"
o:suggestedsigner="John Doe"
o:suggestedsigner2="Manager" issignatureline="t"/>
```

helper.createCTSignatureLine sets **o:suggestedsigner**

You can set other attributes as you see fit (eg the signer's title goes in **o:suggestedsigner2**).

Signing the document

To sign the document, you need a reference to the signature line object. You'll have this if you just inserted the signature line. Otherwise, you'll have to search the document contents to find the signature line (not explained here).

So continuing the example above:

```
SignatureDetail details = helper.configureSignature(fis, password);
details.setVisualSignature(sigLine,
    validSigLnImg png,
    signatureImage png);
helper.sign();
```

validSigLnImg png is critical, its the signature block image (signature plus signature line) which appears on the document surface.

If the signature is invalidated (eg the document is altered), that image will be replaced with **InvalidSigLnImg**. As a convenience, Docx4j can generate that image from **validSigLnImg**, provided you provide it inpng format. Alternatively, you can provide SignatureDetail with your own image to be used if the document is invalid:

```
/**
 * A base64 encoded image (Word uses EMF, but PNG tested ok with Word 2010),
 * displayed in the document if the signature is invalid; made up of the signature line plus
 * the signature, plus red text saying it is invalid.
 *
 * If null, can be generated from validSigLnImg, provided that is a png
 *
 * @param validSigLnImg
 */
public void setInvalidSigLnImg(String invalidSigLnImg)
```

In the Word UI, a user can sign visually with an image, or with text. In the above example, the visual signature is configured with a signature image: *signatureImage png* is an image of the signature; it doesn't appear on the document surface, but is stored in the XmlSignaturePart.

If you wanted to sign instead with text, you'd use:

```
public void setVisualSignature(CTSignatureLine signatureLine, byte[] validSigLnImg,
    String signatureText)
```

USAGE: SIGNATURE VALIDATION

Validate existing signatures:

```
SignatureHelper signing= new SignatureHelper(pkg);
List<XmlSignaturePart> parts = signing.getSignaturesStatus(null);

for (XmlSignaturePart sp : parts) {
    System.out.println(sp.getPartName().getName() + " ---> " + sp.getSignatureStatus(null));
}
```



```
}
```

API SUMMARY

Mostly, you interact with `com.plutext.dsig.SignatureHelper`

Some configuration settings are in `SignatureConfig`. You can access that from `SignatureHelper`:

```
public SignatureConfig getSignatureConfig()
```

configureSignature

```
/**
 * Provide details of a PKCS12 key, in preparation for signing this package
 *
 * @param privateKeyIS
 * @param password
 * @throws Docx4JException
 */
public SignatureDetail configureSignature(InputStream PKCS12stream, String password) throws
Docx4JException

/**
 * Provide details of a PKCS12 key, in preparation for signing this package.
 *
 * @param privateKeyIS
 * @param password
 * @param signatureLine
 * @throws Docx4JException
 */
public SignatureDetail configureSignature(InputStream PKCS12stream, char[] password) throws
Docx4JException

/**
 * Provide details of a key to be used for signing this package
 *
 * @param pkEntry
 */
public SignatureDetail configureSignature(KeyStore.PrivateKeyEntry pkEntry)
```

setVisualSignature

```
/**
 * Set the extra info required to visibly sign this package
 * with a textual signature
 * (associating it with a signature line on the document surface)
 *
 * @param signatureDetail
 * @param signatureLine
 * @param validSigLnImg the image (Word uses EMF, but PNG tested ok with Word 2010
 * and MUST be used since we can't manipulate an EMF, yet),
 * displayed in the document; should be the signature line plus the signature.
 * @param signatureText
 */
public void setVisualSignature(SignatureDetail signatureDetail, CTSignatureLine signatureLine,
byte[] validSigLnImg, String signatureText);

/**
 * Set the extra info required to visibly sign this package
 * with a graphical signature
 * (associating it with a signature line on the document surface)
 *
 * @param signatureDetail
 * @param signatureLine
 * @param validSigLnImg the image (Word uses EMF, but PNG tested ok with Word 2010
 * and MUST be used since we can't manipulate an EMF, yet),
 * displayed in the document; should be the signature line plus the signature.
 * @param signatureImage the signature itself (not actually displayed), but stored
```

```

    */
    public void setVisualSignature(SignatureDetail signatureDetail, CTSignatureLine signatureLine,
        byte[] validSigLnImg, byte[] signatureImage)

```

sign

```

/**
 * Sign it, using the key provided via configureSignature
 */
public void sign() throws Docx4JException

```

miscellaneous

```

/**
 * Signed if SignatureOriginPart has rels to one or more XmlSignatureParts
 * @return
 */
public boolean isPackageSigned()

/**
 * Set the signature status for each signature. This mimics what you see in the Word signatures
pane,
 * and checks the validity of the signature (hash), whether it is complete or partial,
 * and the certificate used to sign with.
 *
 * @param zippedOfficeFile a docx/pptx/xlsx file (not docx4j package), to ensure the input has not
been modified in docx4j
 * @param certificateTrustChecker
 * @return
 * @throws SignaturesNotPresentException
 * @throws DigitalSignatureException
 */
public static List<XmlSignaturePart> getSignaturesStatus(InputStream zippedOfficeFile,
CertificateTrustChecker certificateTrustChecker)
    throws SignaturesNotPresentException, DigitalSignatureException {
    public List<XmlSignaturePart> getSignaturesStatus(CertificateTrustChecker certificateTrustChecker)
throws SignaturesNotPresentException, DigitalSignatureException

```

You can then iterate through the XmlSignaturePart objects, invoking getSignatureStatus(), which returns a SignatureStatus:

```

    public enum SignatureStatus {

        UNKNOWN,
        VALID,
        INVALID,
        VALID_PARTIAL /* the signed parts are valid, but not all required parts are
signed */ ,
        RECOVERABLE /* the signature is valid, but the certificate is not trusted */ ,
        RECOVERABLE_PARTIAL

    }

/**
 * Set the signature status for each signature, without checking the validity of the
 * certificate itself. This is a subset of what you see in the Word signatures pane,
 * since it checks the validity of the signature (hash), and whether it is complete or partial,
 * BUT NOT the certificate used to sign with.
 *
 * @param zippedOfficeFile a docx/pptx/xlsx file (not docx4j package), to ensure the input has not
been modified in docx4j
 * @param certificateTrustChecker
 * @return
 * @throws SignaturesNotPresentException
 * @throws DigitalSignatureException
 */
public static List<XmlSignaturePart> getSignaturesStatusTrusted(InputStream zippedOfficeFile)
throws SignaturesNotPresentException, DigitalSignatureException

/**
 * Remove all signature parts from this package

```

```

* @param pkg
*/
public void removeSignatureParts(OpcPackage pkg)

```

LIMITATIONS

Visible Signatures

1. Visible signatures are currently only supported for Docx files. (Excel 2010 has signature line support, but Docx4j does not currently have a high level API for creating an Excel signature line; please contact Plutext if you need this)
2. Docx4j does not currently support generating a custom signature block EMF image for a user to use to later sign. However, this does not matter if signing is done later via the Office user interface, since Office generates that image itself on-the-fly. It may matter if signing is done via some third party user interface.

Where Docx4j is used to sign the document with a visible signature, the image provided for the signed signature block will appear on the document surface.

Certificate Validation (should a signature be trusted?)

It is possible to examine the signer to determine whether it is to be trusted (based on trusted certificate authorities). This is called "certificate validation", and Word's signature implementation does this.

Docx4j's SignatureHelper contains:

```

/**
 * Set the signature status for each signature. This mimics what you see in the Word signatures
 * pane, and checks the validity of the signature (hash), whether it is complete or partial,
 * and the certificate used to sign with.
 *
 * @param certificateTrustChecker pass null to use the default implementation
 * @return
 * @throws SignaturesNotPresentException
 * @throws DigitalSignatureException
 */
public List<XmlSignaturePart> getSignaturesStatus(CertificateTrustChecker certificateTrustChecker)
throws SignaturesNotPresentException, DigitalSignatureException {

```

You can then invoke `getSignatureStatus(signatureConfig)` on an `XmlSignaturePart` to get its status.

The details of how check the validity of a certificate in Java depends on your environment (including not just whether you use Oracle or IBM java, but if Oracle java, which version, and what certificates you want to trust).

Because of this variation, the API accepts an object implementing:

```

public interface CertificateTrustChecker {

    CertificateTrustCheckResult isTrusted(X509Certificate signer) throws DigitalSignatureException,
    CertPathValidatorException, CertPathBuilderException;

}

```

An implementation of this is provided in `DefaultCertificateTrustChecker`, however it requires Java 8.

If you pass a null `CertificateTrustChecker`, the certificate will not be checked, and providing the signature is valid, the resulting `SignatureStatus` will be `RECOVERABLE` or `RECOVERABLE_PARTIAL`.

X.509 certificate

The X.509 certificate for validating the signature is located in the Digital Signature XML Signature part itself.

In contrast, the Open Packaging Conventions also supports:

- placing it in a Digital Signature Certificate part
- storing the certificate as a separate part in the package, or
- not including it in the package if certificate data is known or can be obtained from a local or remote certificate store.

DSA and RSA algorithms

The Open Packaging Convention spec / Microsoft Office supports both DSA and RSA algorithms

In Docx4j, currently only RSA is supported for signing the digest (see SignatureInfo class).

KNOWN ISSUES WITH OFFICE

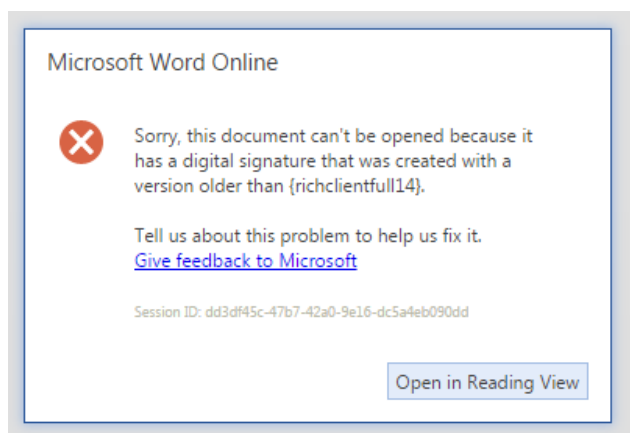
Windows Explorer

If you have the signed docx open in the preview pane in Windows Explorer, Word may be unable to open that docx. So either turn off the preview pane, or preview some other file.

This seems to affect 64 bit versions of Word on Windows 7, 8 and 10.

Word Online

Word Online can't open signed documents (even if signed in eg Word 2013):



(the error message is misleading)

Word for Mac

Word 2011 for Mac is silent as to whether signatures are present.

Word 2016 for Mac detects that a docx is signed, but doesn't give any information about the signature.

Word for Android

Word 1.0.1 (16.0.4027.1006) can't open digitally signed files.

This is fixed in Word 1.0.1 (16.06828.1002), which can open them (but can't validate them).

TROUBLESHOOTING

Validation errors

If signature validation is failing, it is helpful to understand where.

To see, turn on slf4j debug-level logging for: `org.apache.jcp.xml.dsig`

The following VM arguments can also provide insight:

- Dcom.sun.net.ssl.checkRevocation=false
- Dcom.sun.security.enableCRLDP=false
- Djava.security.debug=certpath
- Djavax.net.debug=all

MergePptx

INTRODUCTION

MergePptx is a utility for concatenating pptx presentations together.

It can be used to "re-brand" presentations to have the same look and feel ("theme") as the first one.

Alternatively (since v3.2.0.3), you can keep each presentation's existing theme.

In principle, it could be used to remove slides from a presentation, but there are more efficient ways of doing that.

USAGE

A `SlideRange` represents the slides in a source pptx to be merged.

The simplest constructor says to use the entire presentation (ie all the slides):

```
public SlideRange(PresentationMLPackage pmlPkg) throws MergePptxException
```

Other constructors are discussed below.

Concatenating several entire pptx

To do this:

- construct a `PresentationBuilder` object
- create a `SlideRange` for each pptx, and add it, with

```
builder.addSlideRange(sr);
```

- get the resulting `PresentationMLPackage` using

```
builder.getResult();
```

The result is a new `PresentationMLPackage` containing the contents of the source decks.

For example:

```
String[] deck = {"deck1.pptx" , "deck2.pptx"};

PresentationBuilder builder = new PresentationBuilder();

for (int i=0 ; i< deck.length; i++) {

    // Create a SlideRange representing the slides in this pptx
    SlideRange sr = new SlideRange(
        (PresentationMLPackage)OpcPackage.Load(
            new File(DIR_IN + deck[i])));
    sr.setName(i+ " " + deck[i]); // PkgIdentifier for ListeningBean

    // Add the slide range to the output
    builder.addSlideRange(sr);
}

PresentationMLPackage result = builder.getResult();
```

If you want to use a particular pptx morethan once, you should use clones for each of the subsequent times.

The samples directory contains an example called MergeWholePresentations.

Note: versions earlier than v3.2.0 suggested:

```
List<PresentationMLPackage> pmlPkgs = ...
PresentationMLPackage result = PresentationBuilder.merge(pmlPkgs);
```

however, that requires each input pptx to be in memory at the same time.

Accordingly, that approach is now deprecated.

Concatenating parts of several pptx

If you wish to use only a certain slides, you use the SlideRange constructor:

```
/**
 * @param pmlPkg
 * @param slideNumber 0-based
 * @throws MergePptxException
 */
public SlideRange(PresentationMLPackage pmlPkg,
                  int[] slideNumber) throws MergePptxException
```

For example:

```
PresentationBuilder builder = new PresentationBuilder();

int[] range1 = {2,4,6,8};
builder.addSlideRange( new SlideRange(
    (PresentationMLPackage)OpcPackage.Load(new File(DIR_IN + "microsoft1.pptx")),
    range1) );

int[] range2 = {1,3,5};
builder.addSlideRange( new SlideRange(
    (PresentationMLPackage)OpcPackage.Load(new File(DIR_IN + "oracle.pptx")),
    range2) );

PresentationMLPackage result = builder.getResult();
```

If you want to use a particular pptx in morethan one SlideRange, you should use clones for each of the subsequent times.

The result is a new `PresentationMLPackage` containing the contents of the source decks.

The samples directory contains an example called MergeListedSlides.

Other SlideRange constructors

```
/**
```

```

    * Specify the source package, from "start" (0-based index) to the end of the document
    * @param wordmlPkg
    * @param start
    * @throws MergePptxException
    */
    public SlideRange(PresentationMLPackage pmlPkg, int start) throws MergePptxException

    /**
     * Specify the source package, from "start" (0-based index) and include "count"
     * slides
     *
     * @param wordmlPkg
     * @param start
     * @param count
     * @throws MergePptxException
     */
    public SlideRange(PresentationMLPackage pmlPkg, int start, int count) throws
MergePptxException

```

Deleting part of a pptx

The easiest way to delete slides is to use:

```
public SlideRange(PresentationMLPackage pmlPkg, int[] slideNumber)
```

putting all the slides you want to keep in the array.

SETTINGS

Sections

PowerPoint 2010 introduced the concept of a "section".

In Docx4j Enterprise v8.1.0, when merging presentations, you can preserve existing sections. To do this, set property:

```
com.plutext.merge.pptx.SectionAware=true
```

If that is set to false, sections will not be defined in the merged pptx.

In earlier versions of Docx4j Enterprise, there was no specific section handling.

ThemeTreatment

(since v3.2.0.3)

In a pptx, each slide has a slide layout part.

Each slide layout:

- has a name
- uses a slide master.

And each slide master is linked to a theme.

- A theme has a name, but a slide master doesn't.

By default, MergePptx will use the theme(s) defined in the first presentation. So a slide in a subsequent presentation which uses layout "Title and Content" defined in the first presentation, will take on that appearance.

If a slide uses a layout name not previously encountered (likely a custom layout name), that layout will be added, and associated with an existing theme.

You can use `ThemeTreatment.RESPECT` to have the theme names act as a namespace:

```
PresentationBuilder builder = new PresentationBuilder();
builder.setThemeTreatment(ThemeTreatment.RESPECT);
```

In this case, a slide in a subsequent presentation which uses layout "Title and Content" defined in the first presentation, will only take on the appearance of the earlier presentation if they use a theme with the same name. Otherwise, the theme/master and layout will be imported.

The screenshot above is an example of the Slide Master view for a presentation generated using `ThemeTreatment.RESPECT`.

Note that only the name of the theme matters.

New in v8.4.0.0, there is `ThemeTreatment.RENAME` which is like existing value `RESPECT`, except the Theme names in each deck are made unique. This is useful where the themes have same name (but different content), so user doesn't need to worry about ensuring the names are unique.

If you want to see how your input is being processed, set logger:

```
<logger name="com.plutext.merge.pptx.ThemeTreatment">
  <level value="info"/>
</logger>
```

(and set other loggers to warn or error). This will produce output like:

```
ThemeTreatment .appendSlideRange line 515 - /ppt/slides/p3_slide1.xml
ThemeTreatment .handleLayout line 582 - Uses layout OPN_02_2003:Blank
ThemeTreatment .handleLayout line 588 - .. which we need to add
ThemeTreatment .importLayout line 698 - Importing master /ppt/slideMasters/slideMaster1.xml
ThemeTreatment .appendSlideRange line 515 - /ppt/slides/p3_slide2.xml
ThemeTreatment .handleLayout line 582 - Uses layout OPN_02_2003:Title Slide
ThemeTreatment .handleLayout line 588 - .. which we need to add
ThemeTreatment .appendSlideRange line 515 - /ppt/slides/p3_slide3.xml
ThemeTreatment .handleLayout line 582 - Uses layout OPN_02_2003:Title and Content
ThemeTreatment .handleLayout line 588 - .. which we need to add
ThemeTreatment .appendSlideRange line 515 - /ppt/slides/p3_slide4.xml
ThemeTreatment .handleLayout line 582 - Uses layout OPN_02_2003:Title and Content
ThemeTreatment .handleLayout line 591 - .. which is already present (in /ppt/slideLayouts/slideLayout211.xml)
```

EVENT MONITORING

Since merging presentations can take some time (depending on the number and complexity of the presentations), the possibility exists of performing the merge in the background, and receiving notification when the job is complete.

The MergeWholePresentations sample contains an example of usage.

As per that example, you need to:

- create a message bus, and tell Docx4jEvent to use it

- define and register/subscribe a listener

This is done as follows:

```
// Creation of message bus
MBassador<Docx4jEvent> bus = new MBassador<Docx4jEvent>(
    BusConfiguration.Default());
// tell Docx4jEvent to use your message bus for notifications
Docx4jEvent.setEventNotifier(bus);

// Define and register/subscribe a listener
ListeningBean listener = new ListeningBean();
bus.subscribe(listener);
```

The samples package contains an example ListeningBean. Note the @Handler annotation.

Docx4j's approach to event monitoring relies on the MBassador library; see further

<https://github.com/bennidi/mbassador>

For another example of monitoring events (docx load, save), please see

<https://github.com/plutext/docx4j/blob/master/src/samples/docx4j/org/docx4j/samples/EventMonitoringDemo.java>

Appendix 1 - Installation

To use the Enterprise Edition, you simply add the Enterprise Edition jar to your project.

Background

Docx4j Enterprise relies on JAXB, and the open source docx4j. For Java 8, there are three JAXB implementations to choose from:

- the “internal” implementation shipped with Oracle’s JDK
- the Reference Implementation,
- EclipseLink MOXy

docx4j comes with your preferred JAXB implementation when you include in your project one and only one of:

- docx4j-JAXB-Internal
- docx4j-JAXB-MOXy
- docx4j-JAXB-ReferenceImp

See further <https://search.maven.org/search?q=docx4j> and <https://www.docx4java.org/downloads.html>

Using Maven

If your project is Maven based, you could mvn install docx4j-Enterprise-MergeDocx- 8.4.11.jar in your local maven repository. From a command line, something like:

```
mvn org.apache.maven.plugins:maven-install-plugin:3.1.1:install-file \
-Dfile=docx4j-Enterprise-MergeDocx-8.4.11.jar
```

(Earlier versions of maven-install-plugin don’t read the embedded pom, which means you have to extract the pom from the jar, and specify it with -Dpomfile=pom.xml)

Once you have the jar in your local maven repository, you can add it as a dependency in the pom.xml in your project.

```
<dependency>
  <groupId>com.plutext</groupId>
  <artifactId>docx4j-Enterprise-MergeDocx</artifactId>
  <version>8.4.11</version>
</dependency>
```

This will pull in all the necessary dependencies (except the JAXB implementation), since these are specified in the pom.xml file in the jar.

As per above, add your preferred JAXB implementation by including one and only one of:

- docx4j-JAXB-Internal
- docx4j-JAXB-MOXy
- docx4j-JAXB-ReferenceImp

The only extra thing you need is a logging implementation (implementing the sl4j v2 api) such as logback-classic or log4j-slf4j2-impl; see the sample code pom for an example.

The pattern shown above can also be followed for MergePptx and OLE.

Maven Dependency Notes

For `TextImageGen` we suggest you install it to your local mvn repository, using the copy provided in the lib dir.

For reference, its pom is at <https://github.com/jcraane/textimagegenerator/blob/master/pom.xml> and its repository is as documented there.

Appendix 2 – LOGGING

Like docx4j (from version 3.0), the Enterprise Edition uses slf4j for logging.

As the slf4j website says:

The Simple Logging Facade for Java (SLF4J) serves as a simple facade or abstraction for various logging frameworks (e.g. `java.util.logging`, `logback`, `log4j`) allowing the end user to plug in the desired logging framework at deployment time.

Please see the slf4j website for further information.

Appendix 3 – .NET environment

Introduction

Docx4j OLE can be used in a .NET environment. For this purpose, it is provided as a set of DLLs which can be referenced from a Visual Studio project.

The DLLs were created from the Java jars, using IKVM.

.NET sample solution

Docx4j-OLE for .NET is shipped with a sample solution you can unzip and start working with (for example, in Visual Studio 2010).

The solution structure is as follows:

`Docx_ole_PDF.cs` consists of a few simple steps:

```
// Load the docx
Console.WriteLine("loading docx");
WordprocessingMLPackage wordMLPackage =
    Plutext.Docx4NET.WordprocessingMLPackageFactory.createWordprocessingMLPackage(
        DIR_IN + "simple.docx");

// Create a byte[] out of the PDF we wish to embed
byte[] pdfBytes = File.ReadAllBytes(DIR_IN + "sample.pdf");

// Helper does the work
PdfOleHelper pdfOleHelper = new PdfOleHelper(wordMLPackage);
pdfOleHelper.embed(pdfBytes);

// Save the result
Plutext.SaveFromJavaUtils.save(wordMLPackage, DIR_OUT + @"OUT_ole.docx");
```

After you've built the solution in Visual Studio, be sure to "Start without Debugging". (If you instead choose Debugging, it is orders of magnitude slower)

In the console window, you should see something like:

```
loading docx
reading pdf
.. read 272919
embedding pdf
saving docx
done; wrote to C:\Users\jharrop\AppData\Local\Temp\
Press any key to continue . . .
```

Open that docx and verify you have an embedded PDF. You can then modify the sample project to suit your purposes.

You'll notice there is a startup time to load the DLLs (and the JAXB Context - essentially, reading all the classes which represent the Open XML spec), so you'll want your process to stay resident so you only incur the startup time penalty once.

Logging

You can and should configure logging of docx4j related events in your application, via:

```
Plutext.Log4jConfigurator.Configure(  
    org.apache.log4j.Level.INFO,  
    DIR_OUT + @"\docx4j-OLE_NET_log.txt",  
    true);
```

Please inspect the logs to verify there are no errors.

GAC

The DLLs have strong names, so you can install them in GAC:

```
>gacutil /i docx4j.dll
```

```
Microsoft (R) .NET Global Assembly Cache Utility. Version 4.0.30319.17929  
Copyright (c) Microsoft Corporation. All rights reserved.
```

```
Assembly successfully added to the cache
```

```
>gacutil /i docx4j-OLE.dll
```

```
Microsoft (R) .NET Global Assembly Cache Utility. Version 4.0.30319.17929  
Copyright (c) Microsoft Corporation. All rights reserved.
```

```
Assembly successfully added to the cache
```

To check version etc information, use the /l argument:

```
>gacutil /l docx4j
```

```
Microsoft (R) .NET Global Assembly Cache Utility. Version 4.0.30319.17929  
Copyright (c) Microsoft Corporation. All rights reserved.
```

```
The Global Assembly Cache contains the following assemblies:  
  docx4j, Version=2.9.9.8, Culture=neutral, PublicKeyToken=caeb68ac682d6c33,  
  processorArchitecture=MSIL
```

```
Number of items = 1
```

```
>gacutil /l docx4j-OLE
```

```
Microsoft (R) .NET Global Assembly Cache Utility. Version 4.0.30319.17929  
Copyright (c) Microsoft Corporation. All rights reserved.
```

```
The Global Assembly Cache contains the following assemblies:  
  docx4j-OLE, Version=1.0.2.8, Culture=neutral, PublicKeyToken=caeb68ac682d6c33,  
  processorArchitecture=MSIL
```

```
Number of items = 1
```

ASP.NET notes

There are certain classloading differences between Java and .NET which must be accomodated, particularly in an ASP.NET environment.

The one you need to accommodate in your code is loading of **Xalan**, symptom for which is `org.apache.xalan.processor.TransformerFactoryImpl` not found.

To solve this add to your .NET code:

```
java.lang.Class clazz = typeof(org.apache.xalan.processor.TransformerFactoryImpl);
java.lang.Thread.currentThread().setContextClassLoader(clazz.getClassLoader());
```

or better:

```
ikvm.runtime.Startup.addBootClassPathAssembly(
    System.Reflection.Assembly.GetAssembly(
        typeof(org.apache.xalan.processor.TransformerFactoryImpl)));
```

The latter is better because it is a global setting you only need to do once (at the start of your application). In comparison, `Thread.setContextClassLoader()` is a per thread thing, so it depends on random ASP.NET threading behavior, and, for a given thread, whether any of the Java classes set it.

For more details:

- [IKVM Weblog - Class Loading Architecture](#)
- [IKVM Wiki - Class Loader](#)

Recreating the DLLs

If you have made changes to the Java source code for docx4j OLE Helper (or one of its dependencies, such as docx4j), and you wish to use these changes in .NET, you'll need to regenerate the DLLs.

Steps:

1. **IKVM**: Download IKVM 8.1.5717.0 and install it.
2. **docx4j DLL**: Locate your docx4j DLL (which contains docx4j and all its dependencies); it is included in the docx4j OLE .NET distribution.
If you wish to create the docx4j DLL, then from a dir containing docx4j and all its dependencies (and with ikvmc on your path):
 `> ikvmc -out:docx4j-XX.dll -target:library *.jar`
or, just run docxj's dist.NET ant task (after first setting the ikvm.dir property in build.xml)
Some warnings are expected. The resulting DLL should be approx 24MB.
3. **docx4j-OLE DLL**: In the docx4j-OLE distribution, run the dist.NET ant task (after first setting the ikvm.dir property in build.xml, and the docx4j.dll property to point to DLL from step 2 above).
The resulting DLL will be 7-8MB in size.
4. **Test** the resulting DLLs in the .NET sample project, by replacing the references to the existing DLLs, with references to your new ones, then running the sample.

Troubleshooting. If Visual Studio says it can't load your DLL, look for Fusion Log Viewer (x64) for example at the following location "C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A\Bin\NETFX 4.0 Tools\x64\FUSLOGVW.exe", run it as administrator, click on settings, and enable it, for example to "log in exception text". See further, <http://stackoverflow.com/questions/255669/how-to-enable-assembly-bind-failure-logging-fusion-in-net>