Word to TEI

This is a sample document to test conversion of word docx to TEI using the TEI processing model.

# Style Conversion

Rather than trying to convert everything, the default ODD transformation attempts to preserve the semantics of the text. Most style properties are thus ignored. This is by intention: trying to preserve as much as possible would likely just add noise and result in low-quality TEI.

Users are free to extend the default ODD with additional heuristics. For example, a paragraph being entirely bold could also be treated as a heading, or a left text indent may indicate a quote.

## Inline Styles

Text styled as italic, underline or bold will be transformed into a tei:hi with a corresponding @rend attribute.

Inline styles whose name starts with „tei:“ are transformed into TEI elements with the same name. So if a character sequence uses a style called „tei:persName”, it will be wrapped into a TEI persName element in the output, e.g. Johann Wolfgang Goethe. A place name can be marked with a style „tei:placeName” and should be transformed accordingly: Frankfurt, Berlin, München. And damaged text could be encoded by applying a style „tei:supplied“.

## Paragraph Styles

### Headings

Word does not have a concept for text division, so we have to reconstruct them:

1. Paragraph styles starting with „heading“, „title“ or „subtitle“ generate a tei:head. The outline level assigned to the heading is recorded as well.
2. In a second pass through the generated output, divs are generated based on the outline level:
   1. A div spans all text from the heading to the next heading on the same outline level
   2. Repeat the process for all headings within this division on a lower outline level

### Footnotes

Footnotes are translated into TEI note elements[[1]](#footnote-1). We also support endnotes[[2]](#endnote-1), which are transformed into a note with type=“endnote“.

### Lists

Lists are tricky, because Word essentially just stores list items in a flat list. Reconstructing nesting thus requires looking at the list level associated with every item. Simple lists are easy:

* A list item[[3]](#footnote-2)
* Another list item

We can also have numbered lists, which are translated into a <list type=“ordered“>:

1. First item in a numbered list
2. Second item in a numbered list

Nested lists are quite tricky:

* A bullet list item

1. A nested list item
2. Another nested list item
   1. Nested in a nested list
   2. More …

* A *second* numbered list item

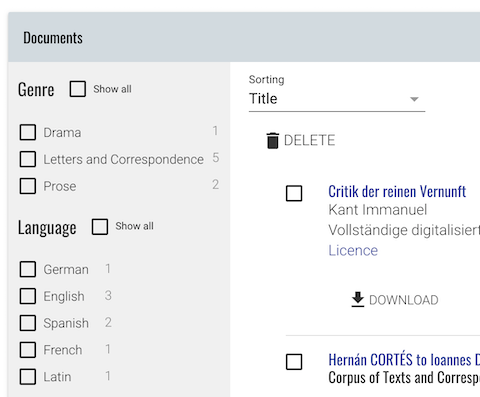
# Tables

We can do simple tables very well. Spanning multiple colums is also easy, but things become more difficult for row spans, which are not implemented yet.

|  |  |  |  |
| --- | --- | --- | --- |
| Item | Hours | Hourly rate | Price |
| Customize ODD | 3 | 120 | 360 |
| Generate App | 4 | 120 | 480 |
| Test and Deploy | 2 | 120 | 240 |
| Total | | | 1080 |

# Embedded Images

Below image will be embedded:



Inside eXist, images are copied into a subcollection starting with the name of the docx file being processed and suffixed with *.media*.

1. A *footnote* may contain inline **formatting**. [↑](#footnote-ref-1)
2. This is an *endnote*, which should appear at the very end of the text. [↑](#endnote-ref-1)
3. And here we have a footnote with a [link](#target1) to another place in the document. [↑](#footnote-ref-2)