Format v1, 2023

Company: Format helps employees automatically style their documents to better match internal design standards.

Objective: The objective of Format v1 is to create a simple web application where a user can upload a .docx source file (with desired styling) and a target .docx file (with no styling or incorrect styling) and automatically apply the styling from the source to the target file.

Background on Office File Formats: The metadata for document styling in Microsoft Office files like .docx, .pptx, etc is stored within XML files within the file's zipped archive structure. The specific XML files for styling information include "styles.xml" for styling information and "document.xml" for content. These files can be edited directly to modify the style information.

Examples of properties in the “styles.xml” file for .docx files:

- Type: paragraph or character
- Style ID: unique identifier for the style
- Name: name of the style
- BasedOn: identifier of the style that this style is based on
- Next: identifier of the style to be used for the next paragraph
- Font: font name, size, color, bold, italic, underline, etc.
- Paragraph: alignment, indentation, spacing, borders, etc.
- Tabs: tab stop positions and types
- Numbering: list numbering and formatting information
- Shading: background color information
- Table: table-specific styles such as table borders and cell shading.

See appendix 1 for an example styles.xml file

Ideal architecture: For simplicity I’d like to use python for this application (see existing lxml library). I am also leaning towards Heroku for app deployment, but open to alternatives during this prototyping stage.

Requirements:

- Priority 0 (must have):
  - Target file must inherit correct font type, font color, font size, paragraph spacing from source file
- Priority 1 (nice to have):
  - Table styles
  - Page margins
- Priority 2 (bonus):
  - Additional styles
  - Any images or text (with styling) that exist in the header and footer of source document

Format v1, 2023

Mapping Styles to Elements and Resolving Conflicts:
A property called “style id” links styles in the styles.xml file to elements in the document.xml file (e.g. "Heading 1", "Title", "Normal", "List Paragraph", "Quote", etc.).

- Acceptable (frequency-based)
    - Apply the style from the source file that is most often used for the corresponding style id. For example, if there are three Title tags in the document and two of them have purple font and a third Title tag has orange font, apply purple font color to Title tags in the target file.
- Ideal (intelligent)
    - Since average document authors do not use distinct elements very often, try to identify the hierarchy of information and implied structure using font-size alone. The heading of the file might not have the Heading 1 tag but will likely still have larger font size than the rest of the document.

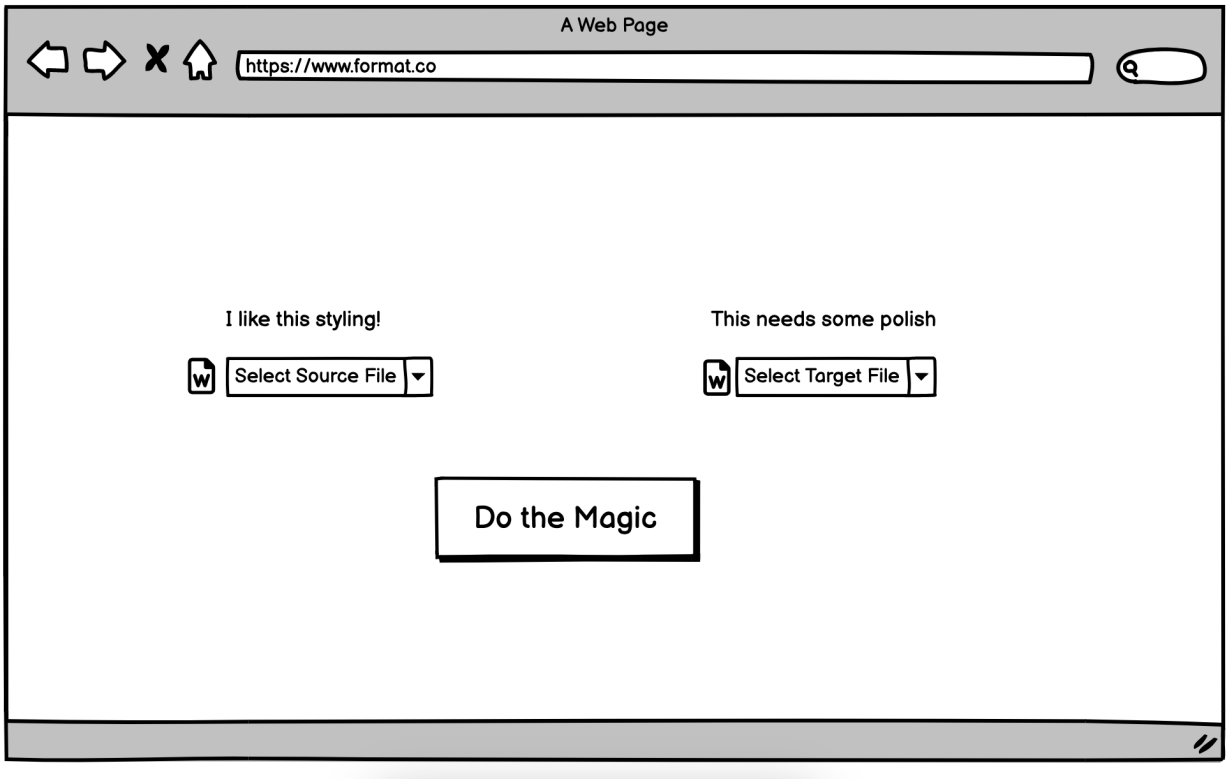
Resources:
- How to use Open XML to extract styles from a Word doc
- About Open XML SDK

Sample Script:
This script reads the contents of the "styles.xml" file within the .docx file and parses it as an XML document using the lxml library. Then, it iterates over all the "w:style" elements in the document, extracting the style ID, type, and name, and printing the results.

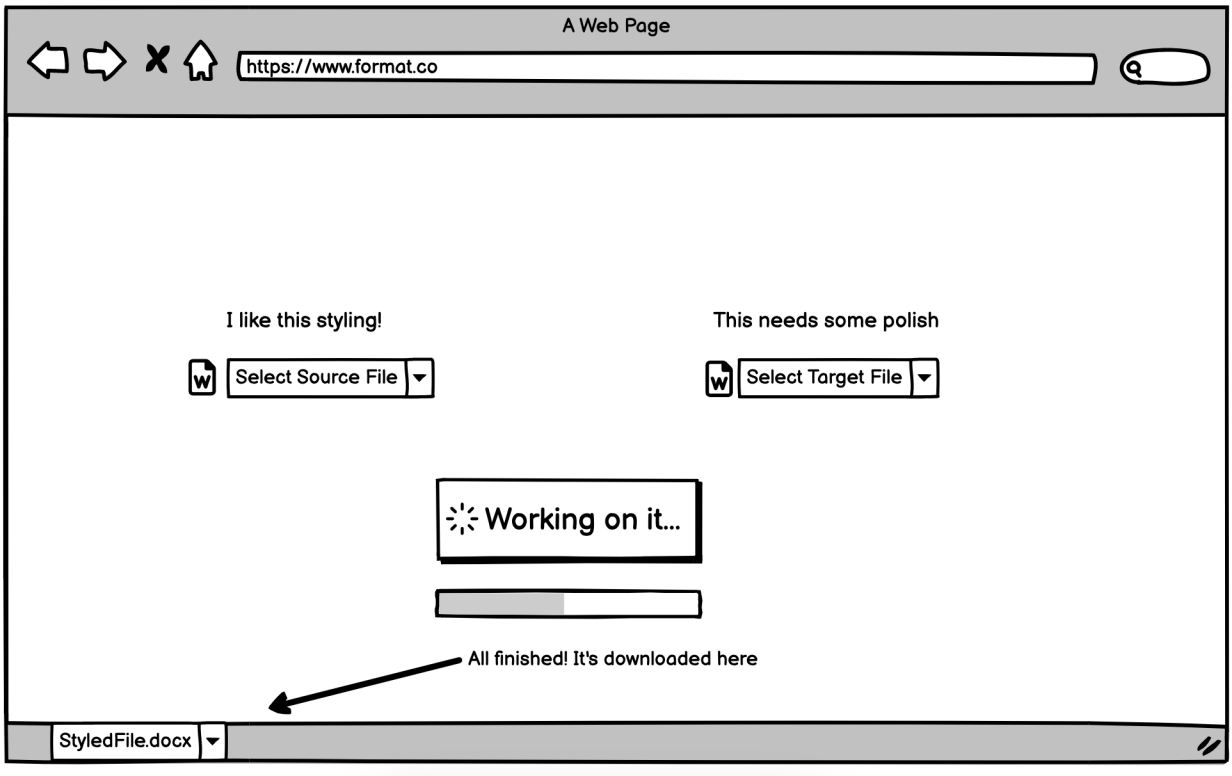
```
import zipfile
import lxml.etree as ET

# Open the .docx file as a ZIP archive
with zipfile.ZipFile("document.docx", "r") as archive:
    # Read the contents of the styles.xml file
    styles_xml = archive.read("word/styles.xml").decode("utf-8")

# Parse the styles.xml contents as an XML document
root = ET.fromstring(styles_xml)

# Iterate over all the w:style elements in the document
for style_element in root.findall(".//w:style", namespaces=root.nsmap):
    # Extract the style ID and type
    style_id =
style_element.attrib["{http://schemas.openxmlformats.org/wordprocessingml/2006/main}styleId"]
    style_type =
style_element.attrib["{http://schemas.openxmlformats.org/wordprocessingml/2006/main}type"]

    # Extract the name of the style
```

```
    name_element = style_element.find(".//w:name", namespaces=root.nsmap)
    style_name =
name_element.attrib["{http://schemas.openxmlformats.org/wordprocessingml/2006/main}val"]

    # Print the style information
    print(f"Style ID: {style_id}")
    print(f"Type: {style_type}")
    print(f"Name: {style_name}")
    print()
```

Expected User Interface



Select both files, click button



Loading message while program runs. Styled target file automatically downloads

Milestones

#	Description	Delivered Date/Time
1	Program can extract font-type and font-color within source file and list those properties	
2	Program can extract font-type and font-color within source file and apply them to target file (assuming all style ids map to correct entities like Header, Title, etc)	
3	Program can extract font-type, font-color, font-size, and paragraph spacing within source file and apply them to target file	
4	Program is hosted and can be run in a web browser	
5	Program can perform everything in 1-4 and also applies styling from tables and page margins	
6	Program can perform everything in 1-5 and also resolves style conflicts using frequency logic (most common value per property is assigned)	

7	Program can perform everything in 1-5 and also resolves style conflicts using intelligent logic (font-size for implied headings) and frequency logic as fallback	
8	Program can do everything in 7 and also transfer additional styling elements (TBD)	Not applicable

Sample Source File and Target File for steps 1 to 3
Sample Source File and Target File for steps 5 to 7

APPENDIX

Appendix 1: Example styles.xml file for .docx file type

```
<w:styles xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main">
  <w:style w:type="paragraph" w:styleId="Normal">
    <w:name w:val="Normal"/>
    <w:basedOn w:val="Normal"/>
    <w:next w:val="Normal"/>
    <w:pPr>
      <w:spacing w:after="200" w:line="276" w:lineRule="auto"/>
    </w:pPr>
  </w:style>
  <w:style w:type="paragraph" w:styleId="Heading1">
    <w:name w:val="Heading 1"/>
    <w:basedOn w:val="Normal"/>
    <w:next w:val="Normal"/>
    <w:pPr>
      <w:spacing w:after="0" w:line="240" w:lineRule="auto"/>
      <w:outlineLvl w:val="0"/>
    </w:pPr>
    <w:rPr>
      <w:b/>
      <w:color w:val="2E74B5"/>
      <w:sz w:val="32"/>
      <w:szCs w:val="32"/>
    </w:rPr>
  </w:style>
  <w:style w:type="character" w:styleId="Emphasis">
    <w:name w:val="Emphasis"/>
    <w:basedOn w:val="Normal"/>
```

```
    <w:rPr>
      <w:i/>
    </w:rPr>
  </w:style>
</w:styles>
```

This example shows three styles defined in the "styles.xml" file. The first style is the "Normal" style, which sets the spacing after a paragraph to 200 twips and line spacing to 276 twips. The second style is "Heading 1", which sets the font size to 32 points and makes the text bold and blue. The third style is "Emphasis", which sets the text in italic.