Aspose.Word

A library for document processing tasks

Written in Java

By: Mobina Goodarzi

Table of Contents

Introduction

Aspose.Words for Java is a class library that enables your applications to perform a great range of document processing tasks.

Aspose.Words supports most of the popular document formats such as DOC, DOCX, RTF, HTML, Markdown, PDF, XPS, EPUB, and others.

With Aspose.Words for Java, you can generate, modify, convert, render, and print documents without third-party applications or Office Automation.

Setting up and using Aspose library in Java

Install Aspose.Words for Java from Maven Repository

Aspose hosts all Java APIs in Maven repository. You can easily use Aspose.Words for Java API directly in your Maven Projects with simple configurations:

1. First, you need to specify Aspose Maven Repository configuration/location in your Maven pom.xml as shown below:

```
<repositories>
	<repository>
		<id>AsposeJavaAPI</id>
		<name>Aspose Java API</name>
		<url>https://releases.aspose.com/java/repo/</url>
	</repository>
</repositories>
```

2. Then, define the Aspose.Words for Java API dependency in your pom.xml as follows:

```
<dependencies>
	<dependency>
		<groupId>com.aspose</groupId>
		<artifactId>aspose-words</artifactId>
		<version>22.11</version>
		<classifier>jdk17</classifier>
	</dependency>
	<dependency>
		<groupId>com.aspose</groupId>
		<artifactId>aspose-words</artifactId>
		<version>22.11</version>
		<classifier>javadoc</classifier>
	</dependency>
</dependencies>
```

Document Creator Class

```java
package org.example;

import com.aspose.words.Document;
import com.aspose.words.DocumentBuilder;
import com.aspose.words.SaveFormat;

1 usage
public class DocumentCreator {
    1 usage
    public static void createWordDocument(String documentPath) throws Exception {
        Document doc = new Document();
        // DocumentBuilder provides members to easily addImage content to a document.
        DocumentBuilder builder = new DocumentBuilder(doc);
        // Write a new paragraph in the document with some text as "Sample Content..."
        builder.writeln( text: "Aspose Sample Content for Word file.");
        // Save the document in DOCX format. The format to save as is inferred from the extension of the file name.
        // Aspose.Words supports saving any document in many more formats.
        doc.save(documentPath, SaveFormat.DOCX);
    }
}
```

This method creates a new Word document.

- It uses the **Document** class constructor to initialize a new document object.
- It creates a **DocumentBuilder** object to facilitate content addition to the document.
- The **DocumentBuilder**'s **writeln** method is used to add sample content to the document.
- Finally, the **save** method of the **Document** class is called to save the document in the DOCX format at the specified path.

Document Editor Class

```
public class DocumentEditor {
    no usages
    public static void accessToTableCells(String documentPath) throws Exception {

        Document doc = new Document(documentPath);

        //Getting information from the first table
        Table table = (Table) doc.getChild(NodeType.TABLE, index: 0, isDeep: true);

        if (table != null) {
            //Get the first cell of first row
            Cell cell = (Cell)table.getRows().get(0).getCells().get(0);

            Paragraph paragraph = cell.getFirstParagraph();
            //cell.getParagraphs().get(0); //or use this
            paragraph.removeAllChildren(); // Delete all content inside the paragraph
            paragraph.appendChild(new Run(doc, text: "New cell content"));
```

DocumentEditor class provides several methods for editing Word documents like;

1. **accessToTableCells(String documentPath)**: This method allows access to the cells of table in the specified Word document. It retrieves the selected cell of the selected row, and applies the desired changes.
2. **addTableToDocument(String documentPath)**: This method adds a table to the specified Word document. It utilizes some table method from the **Tables** class.
3. **addImage(String documentPath)**: This method inserts an image into the specified Word document. It inserts the image located at the desired path.

Tables Class

```
public class Tables {
    1 usage
    public static void formattedTable(DocumentBuilder builder) throws Exception {

        Table table = builder.startTable();
        builder.insertCell();

        // Table wide formatting must be applied after at least one row is present in the formattedTable.
        table.setLeftIndent(20.0);

        // Set height and define the height rule for the header row.
        builder.getRowFormat().setHeight(40.0);
        builder.getRowFormat().setHeightRule(HeightRule.AT_LEAST);

        builder.getCellFormat().getShading().setBackgroundPatternColor(new Color((198), (217), (241)));
        builder.getParagraphFormat().setAlignment(ParagraphAlignment.CENTER);
```

This **Tables** class provides methods to generate various types of tables in Word documents.

These methods offer flexibility in creating tables with different formats and styles.

Existence Checker Of Document Class

```java
package org.example;

import java.io.File;

2 usages
public class ExistenceCheckerOfDocument {
    2 usages
    public static boolean isDocumentExists(String documentPath) {
        File file = new File(documentPath);
        return file.exists();
    }
}
```

The **ExistenceCheckerOfDocument** class contains a static method **isDocumentExists(String documentPath)** that checks whether a document file exists at the specified path. It takes a **String** parameter **documentPath**, which represents the path to the document file, and returns a boolean value.

Font Checker Class

```java
package org.example;

import java.awt.*;
import java.io.File;
import java.io.IOException;

2 usages
public class FontChecker {
    1 usage
    public static boolean isFontInstalled(String fontName) {
        String[] fontNames = GraphicsEnvironment.getLocalGraphicsEnvironment().getAvailableFontFamilyNames();
        for (String name : fontNames) {
            if (name.equals(fontName)) {
                return true;
            }
        }
        return false;
    }

    1 usage
    public static void installFont(String fontFilePath) {
        try {
            GraphicsEnvironment ge = GraphicsEnvironment.getLocalGraphicsEnvironment();
            ge.registerFont(Font.createFont(Font.TRUETYPE_FONT, new File(fontFilePath)));
        } catch (IOException | FontFormatException e) {
            e.printStackTrace();
        }
    }
}

```

The **FontChecker** class contains two static methods:

1. **isFontInstalled(String fontName): boolean**: This method checks if a font with the specified name is installed on the system. It takes a **String** parameter **fontName**, representing the name of the font to check, and returns a boolean value.
2. **installFont(String fontFilePath)**: This method installs a custom font from the specified file path into the system. It takes a **String** parameter **fontFilePath**, representing the path to the font file (TTF format), and installs the font if it's not already installed.

Challenges

Limited free features: Aspose.Words has restrictions on free usage, including the insertion of a message stating "Created with an evaluation copy of Aspose.Words" in documents generated with the evaluation version.

Lack of extensive learning resources: There is a shortage of comprehensive tutorials and examples for Aspose.Words, requiring users to rely primarily on official documentation.

Alternative solution: Apache POI is a popular open-source alternative to Aspose.Words, offering similar functionality for working with Microsoft Office documents in Java, along with extensive documentation and community support.