How to Read/Write Data from Excel in Selenium Using Apache POI?

Overview

Selenium is a widely-used open-source testing framework for web applications. It offers a practical method for automating web browsers and carrying out numerous testing activities. In Selenium test automation, knowing how to read data from Excel in Selenium is a regular requirement. Apache POI (Poor Obfuscation Implementation) is a popular Java API that provides support for reading and writing Microsoft Office file formats, including Excel.

What is Apache POI?

Apache POI (Poor Obfuscation Implementation) is a powerful Java API developed by the Apache Software Foundation. It offers assistance with Excel, Word, and PowerPoint file types as well as reading, writing, and editing other Microsoft Office file formats. To generate, alter, and extract data from Office documents, Java developers can use Apache POI to communicate programmatically with various file formats.

apache poi

Since working with Excel files is Apache POI's primary goal, it is a popular option for data-driven testing in automation frameworks like Selenium. It gives developers access to a wide range of classes and interfaces that abstract away the underlying file formats and lets them manipulate Excel workbooks, worksheets, rows, and cells in a variety of ways.

Getting Started with Apache POI

Before we dive into how to read data from Excel in Selenium using Apache POI, let's get started by setting up the necessary dependencies and understanding the basic concepts of Apache POI.

To begin, we need to add the Apache POI dependency to our project. Apache POI provides different artifacts for working with different Office formats. For Excel manipulation, we need to include the following Maven dependency:

```xml
<dependency>
   <groupId>org.apache.poi</groupId>
   <artifactId>poi</artifactId>
   <version>4.1.2</version>
</dependency>
```

Once we have added the dependency, we can start using the classes and interfaces provided by Apache POI.

Classes and Interfaces in Apache POI
classes and interfaces in apache poi

Apache POI offers a rich set of classes and interfaces to work with Excel files. Here are some of the key ones:

Workbook: The Workbook class represents an Excel workbook, which serves as the main entry point in how to read Excel data from Selenium Driver using Java's Apache POI. It provides methods to create, access, and manage sheets within the workbook. For various Excel file types, Apache POI provides various Workbook interface implementations, such as HSSFWorkbook for.xls files and XSSFWorkbook for.xlsx files.

Sheet: The Sheet class represents a worksheet within a workbook. It allows you to access and manipulate data within a specific sheet. With Sheet, you can perform operations like creating rows and cells, retrieving cell values, applying to format, and managing sheet-level properties such as name, visibility, and order.
Row: A row on a sheet is represented by the Row class. You can use it to gain access to and manipulate the row's cells. The Row class allows you to construct cells, obtain and set cell values, and format cells inside of a row. Additionally, it offers ways to control row-level attributes like height and visibility.
Cell: A single cell in a row is represented by the Cell class. It offers ways to access and modify cell values, format cells, and apply formulas to them. With Cell, you can handle cell-specific attributes like borders, alignment, and data validation as well as get and set cell data types, format cell values according to their data type, apply styles to cells, and more.
DataFormatter: The DataFormatter class is a utility class provided by Apache POI to help format cell values based on their data type. It automatically applies the appropriate formatting for different types of cell values, such as dates, numbers, and booleans. This class is particularly useful when reading cell values to ensure consistent formatting and avoid data type mismatches.
FileInputStream: The Java I/O API's FileInputStream class is used in conjunction with Apache POI to read data from Excel in Selenium. By giving Excel files an input stream, which can then be sent to the necessary Apache POI classes for additional processing, it makes it possible to read Excel files.
FileOutputStream: The FileOutputStream class is used to write data to a file, much like FileInputStream is. By giving the file an output stream, it makes it possible to write Excel data and save it in the chosen file location using Apache POI.
Read Data from Excel in Selenium
To read data from excel sheet in selenium webdriver using java Apache POI, you need to follow a series of steps. Here's a step-by-step explanation:

1. Create an Instance of the Workbook Class and Open the Excel File
Code

```
FileInputStream file = new FileInputStream("path/to/excel/file.xlsx");
Workbook workbook = new XSSFWorkbook(file);
```

Explanation In this step, you create a FileInputStream object to read the Excel file from the specified path. Then, you create an instance of the Workbook class, specifically the XSSFWorkbook implementation for .xlsx files, and pass the FileInputStream object as a parameter to open the workbook.

2. Get the Desired Sheet from the Workbook.
Code

```
Sheet sheet = workbook.getSheet("Sheet1");
```

Explanation Here, you use the getSheet() method of the Workbook class to retrieve the desired sheet by specifying its name. You can replace Sheet1 with the actual name of the sheet you want to read data from.

3. Iterate Over the Rows and Cells to Read the Data.
Code

```
Iterator<Row> rowIterator = sheet.iterator();
while (rowIterator.hasNext()) {
  Row row = rowIterator.next();
```

```
    Iterator<Cell> cellIterator = row.iterator();
    while (cellIterator.hasNext()) {
        Cell cell = cellIterator.next();
        // Process the cell value
    }
}
```

Explanation In this step, you create an iterator for the rows in the sheet using the iterator() method of the Sheet class. You then iterate over each row using a while loop and retrieve the Row object. Inside the row loop, you create an iterator for the cells within the row and iterate over them using another while loop. For each cell, you retrieve the Cell object.

4. Process the Cell Value Based on its Data Type using the DataFormatter.
Code

```
DataFormatter dataFormatter = new DataFormatter();
String cellValue = dataFormatter.formatCellValue(cell);
```

Explanation To handle different data types of cell values, you create an instance of the DataFormatter class. The DataFormatter class helps in formatting cell values based on their data type. You use the formatCellValue() method of the DataFormatter class to get the formatted value of the cell as a String.

5. Close the Workbook and File Input Stream.
Code

```
workbook.close();
file.close();
```

Explanation Finally, after reading the data, you need to close the workbook and the FileInputStream to release system resources.

Remember to handle exceptions and perform appropriate error handling as necessary. You can add try-catch blocks to handle IOExceptions that may occur during file operations.

By following these steps, you can successfully read data from an Excel file in Selenium using Apache POI. You can process the cell values as per your testing requirements, such as using them as test data or validating against expected values in your test automation scripts.

Write Data into Excel in Selenium
To write data to an Excel file in Selenium using Apache POI, you can follow these steps. Here's a step-by-step explanation

1. Create an Instance of the Workbook Class and Create a New Excel File.
Code

```
Workbook workbook = new XSSFWorkbook();
Sheet sheet = workbook.createSheet("Sheet1");
```

Explanation In this step, you create a new instance of the Workbook class, specifically the XSSFWorkbook implementation for .xlsx files. This creates a new in-memory workbook object. Then, you create a new sheet within the workbook using the createSheet() method and provide a name for the sheet, such as Sheet1.

2. Create Rows and Cells and Write the Data.
Code

```
Row row = sheet.createRow(0);
cell.setCellValue("Hello, World!");
```

Explanation Here, you create a Row object within the sheet using the createRow() method, passing the row index as the parameter. In this example, we create a row at index 0. Then, you create a Cell object within the row using the createCell() method and pass the cell index as the parameter.

In this case, we create a cell at index 0 of the row. Finally, you set the cell value using the setCellValue() method. You can provide any desired value, such as Hello, World!.

3. Create a File Output Stream and Write the Workbook Data to a File.
Code

```
FileOutputStream file = new FileOutputStream("path/to/excel/file.xlsx");
workbook.write(file);
```

Explanation In this step, you create a FileOutputStream object to specify the file where you want to write the data. You provide the file path as a parameter. Then, you use the write() method of the Workbook class to write the workbook data to the file specified by the FileOutputStream object.

4. Close the Workbook and File Output Stream.
Code

```
workbook.close();
file.close();
```

Explanation Finally, after writing the data, you need to close the workbook and the FileOutputStream to release system resources.

Remember to handle exceptions and perform appropriate error handling as necessary. You can add try-catch blocks to handle IOExceptions that may occur during file operations.

By following these steps, you can successfully write data to an Excel file in Selenium using Apache POI. This can be useful for scenarios such as populating test data, generating reports, or creating data-driven test cases in your automation scripts.