# Data Table PDF Generator
## User Manual

## Table of Contents

## INTRODUCTION

PDF utility JAR will generate PDF document and write multipage data table into pdf document from the given standard input. Users can just download below JAR and place it in their CLASSPATH.
Download PDFUtil Jar.

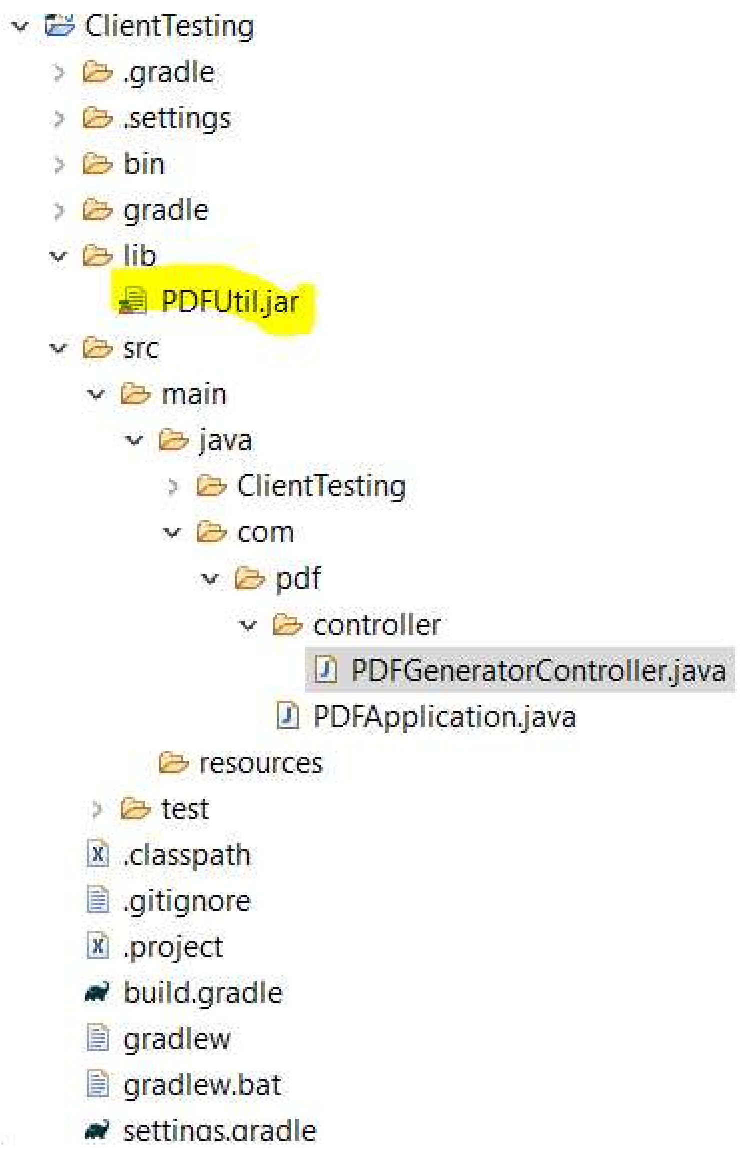
PDFUtil.jar

PDFUtil is simple Java Archive file contains wrapper class built using Apache PDFBox® library. The Apache PDFBox® library is an open source Java tool for working with PDF documents. This utility provides a level abstraction where data provided in format data-table can be directly use to generate PDF documents, which further reduces overall complexity, and underlying boilerplate code. It has several features including:

1. Users can create empty PDF page.
2. Allow users to write text to created PDF document.
3. Allow users to draw line in the document.
4. Users can convert the document to byte stream.
5. Users can generate the multipage Table from user input.

## INSTRUCTIONS

1. Download above JAR and place it in your class path.

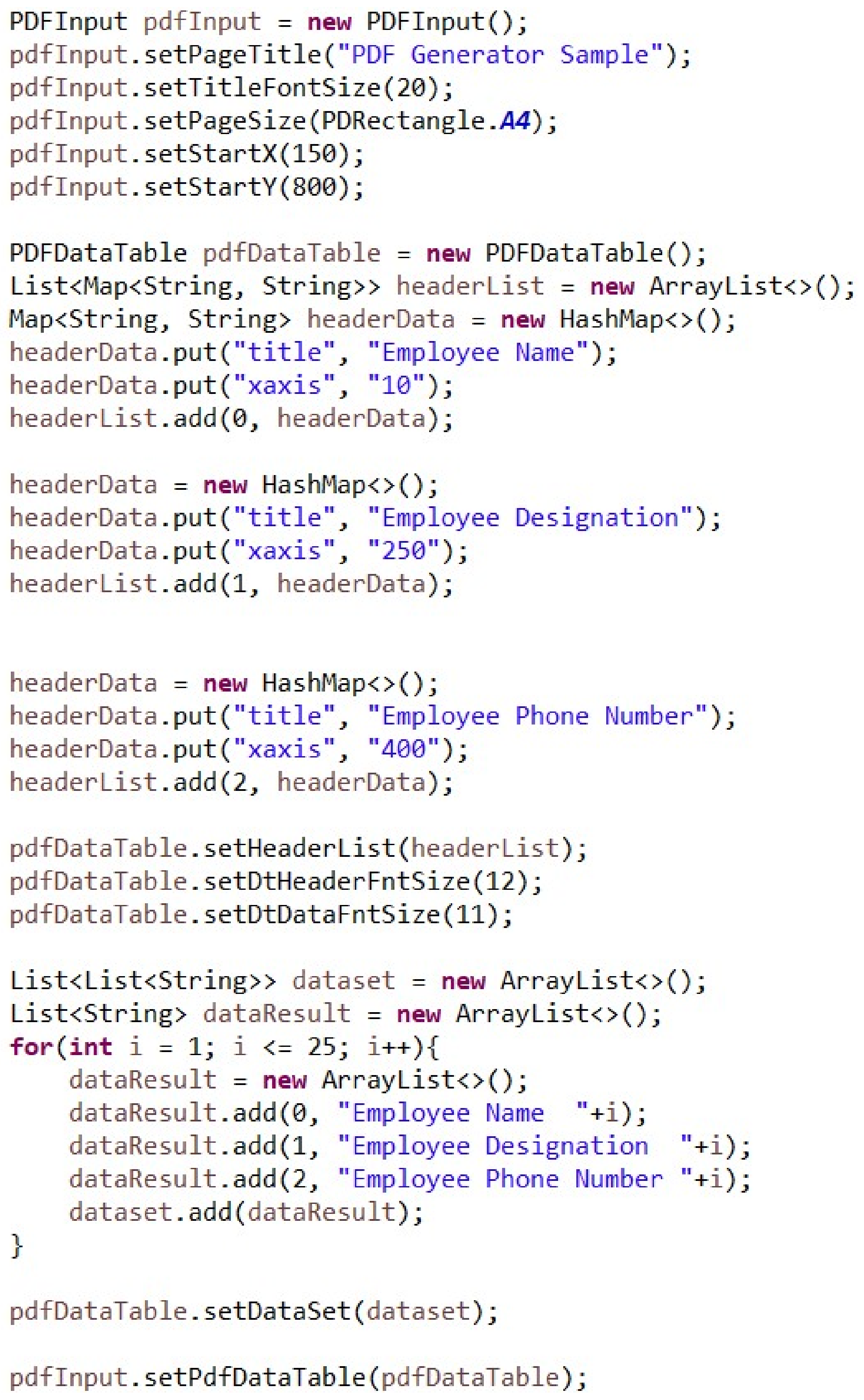


```
ClientTesting
  .gradle
  .settings
  bin
  gradle
  lib
    PDFUtil.jar
  src
    main
      java
        ClientTesting
        com
          pdf
            controller
              PDFGeneratorController.java
            PDFApplication.java
      resources
    test
  .classpath
  .gitignore
  .project
  build.gradle
  gradlew
  gradlew.bat
  settings.gradle
```

Note:- If GRADLE build, please add below line in your script.

compile fileTree(dir: 'lib', include: '*.jar')

2. Create a PDFInput class object and fill the required details.

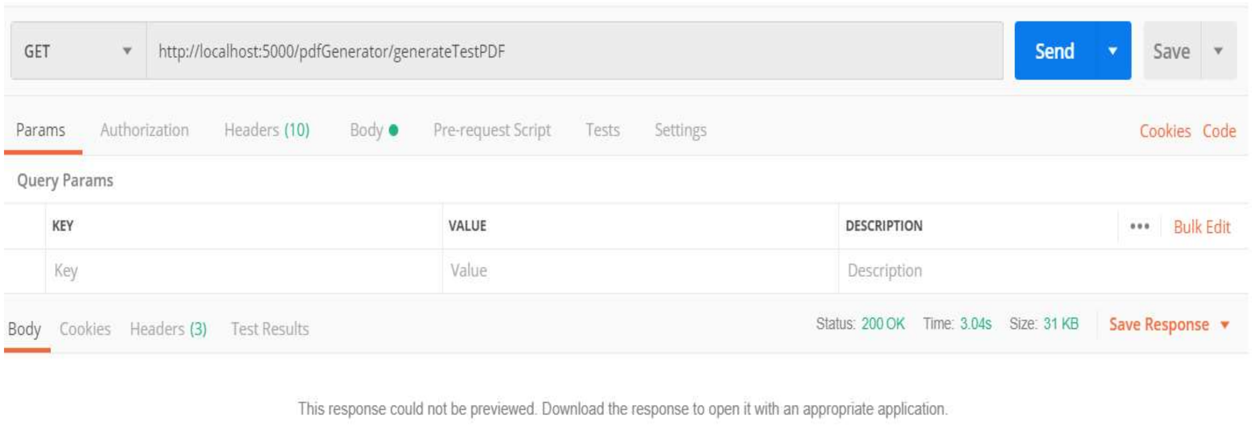


```java
PDFInput pdfInput = new PDFInput();
pdfInput.setPageTitle("PDF Generator Sample");
pdfInput.setTitleFontSize(20);
pdfInput.setPageSize(PDRectangle.A4);
pdfInput.setStartX(150);
pdfInput.setStartY(800);

PDFDataTable pdfDataTable = new PDFDataTable();
List<Map<String, String>> headerList = new ArrayList<>();
Map<String, String> headerData = new HashMap<>();
headerData.put("title", "Employee Name");
headerData.put("xaxis", "10");
headerList.add(0, headerData);

headerData = new HashMap<>();
headerData.put("title", "Employee Designation");
headerData.put("xaxis", "250");
headerList.add(1, headerData);


headerData = new HashMap<>();
headerData.put("title", "Employee Phone Number");
headerData.put("xaxis", "400");
headerList.add(2, headerData);

pdfDataTable.setHeaderList(headerList);
pdfDataTable.setDtHeaderFntSize(12);
pdfDataTable.setDtDataFntSize(11);

List<List<String>> dataset = new ArrayList<>();
List<String> dataResult = new ArrayList<>();
for(int i = 1; i <= 25; i++){
    dataResult = new ArrayList<>();
    dataResult.add(0, "Employee Name  "+i);
    dataResult.add(1, "Employee Designation  "+i);
    dataResult.add(2, "Employee Phone Number "+i);
    dataset.add(dataResult);
}

pdfDataTable.setDataSet(dataset);

pdfInput.setPdfDataTable(pdfDataTable);
```

3. Call the PDFUtil generateDataTablePDF method to get the PDF byte stream data.

```
pdfInput.setPdfDataTable(pdfDataTable);
try {
    pdfBytes = PDFUtil.generateDataTablePDF(pdfInput);
} catch (IOException e1) {
    e1.printStackTrace();
}
response.setContentType("application/pdf");
OutputStream out = null;
try {
    out = response.getOutputStream();
    out.write(pdfBytes);
} catch (IOException e) {
    e.printStackTrace();
}
```

4. Above is sample spring boot application created for testing, where PDFUtil JAR is used to generate PDF document.



GET http://localhost:5000/pdfGenerator/generateTestPDF

Query Params

| KEY | VALUE | DESCRIPTION |
|---|---|---|
| Key | Value | Description |

Status: 200 OK Time: 3.04s Size: 31 KB

This response could not be previewed. Download the response to open it with an appropriate application.

5. Please see below code snippet and sample output file for reference.

SAMPLE CODE SNIPPET

```
byte[] pdfBytes = null;

PDFInput pdfInput = new PDFInput();
pdfInput.setPageTitle("PDF Generator Sample");
pdfInput.setTitleFontSize(20);
pdfInput.setPageSize(PDRectangle.A4);
pdfInput.setStartX(150);
pdfInput.setStartY(800);

PDFDataTable pdfDataTable = new PDFDataTable();

List<Map<String, String>> headerList = new ArrayList<>();
Map<String, String> headerData = new HashMap<>();
headerData.put("title", "Employee Name");
headerData.put("xaxis", "20");
headerList.add(0, headerData);

headerData = new HashMap<>();
headerData.put("title", "Employee Designation");
headerData.put("xaxis", "250");
headerList.add(1, headerData);

headerData = new HashMap<>();
headerData.put("title", "Employee Phone Number");
headerData.put("xaxis", "400");
headerList.add(2, headerData);

pdfDataTable.setHeaderList(headerList);
pdfDataTable.setDtHeaderFntSize(12);
pdfDataTable.setDtDataFntSize(11);

List<List<String>> dataset = new ArrayList<>();
List<String> dataResult = new ArrayList<>();
for(int i = 0; i < 10; i++){
	dataResult = new ArrayList<>();
	dataResult.add(0, "Employee Name "+i);
	dataResult.add(1, "Employee Designation "+i);
	dataResult.add(2, "Employee Phone Number "+i);
	dataset.add(dataResult);
}

pdfDataTable.setDataSet(dataset);

pdfInput.setPdfDataTable(pdfDataTable);
try {
	pdfBytes = PDFUtil.generateDataTablePDF(pdfInput);
} catch (IOException e1) {
	e1.printStackTrace();
}
```

```
        response.setContentType("application/pdf");
        OutputStream out = null;
        try {
            out = response.getOutputStream();
            out.write(pdfBytes);
        } catch (IOException e) {
            e.printStackTrace();
        }
```

SAMPLE OUTPUT FILE:

response.pdf

## API SPECIFICATIONS

### PDFUtil

Package: com.pdf

Description: This class have various static methods to generate PDF and to write the text to it.

### PDFInput

Package com.pdf.pojo

Description: This class is input to the PDF Util class which contains basic required variable for PDF document creation.

### PDFDataTable

Package com.pdf.pojo

Description: This class is one of the variable in PDFInput class. This class have all the requirement member variable for datatable creation.

### generateDataTablePDF

*Method signature:* generateDataTablePDF(PDFInput pdfInput)

*Return Type:* byte[]

*Description:* This method will generate multipage table structure based on User Input and return the create pdf byte stream.

*Sample:* PDFUtil.generateDataTablePDF(pdfInput);

## addPage

*Method signature*: addPage(PDDocument document, PDFInput pdfInput)

*Return Type:* PDPageContentStream

*Description:* This method will create page and return content stream.

## generateDataTable

*Method signature:* generateDataTable(PDDocument document, PDPageContentStream contentStream, PDFInput pdfInput)

*Return Type:* float

*Description:* This method will create data table at specified location on given document and return the current location

## addText

*Method signature*: addText(PDPageContentStream contentStream, float startX, float startY, String text, float fontSize)

*Return Type:* Void

*Description:* This method will add text at specified location on given document

## addBoldText

*Method signature:* addBoldText(PDPageContentStream contentStream, float startX, float startY, String text, float fontSize)

*Return Type:* Void

*Description:* This method will add BOLD text at specified location on given document

## addParagraph

*Method signature:* drawline(PDPageContentStream contentStream, float width, float startX, float startY)

*Return Type:* Void

*Description:* This method will add paragraph at specified location on given document.

## drawline

*Method signature:* drawline(PDPageContentStream contentStream, float width, float startX, float startY)

*Return Type:* Void

***Description:*** This method will draw line at specified location on given document stream.

## addSplitText

***Method signature***: addSplitText(PDPageContentStream contentStream, float startX, float startY, String text, float fontSize, int wrapTextLen)

***Return Type:*** Void

***Description:*** This method will write multi line at specified location on given document stream and return the new Y location.