Chapter 4. Development on ABAP and JAVA Stacks

SAP PI is based on both ABAP and JAVA stacks, and this chapter concentrates on development done on both stacks in both languages – ABAP and JAVA. This is one of the reasons why every SAP Process Integration developer needs to know how to work with both languages. The best way to become familiarized with ABAP and JAVA development for SAP PI is to review real life scenarios that require related knowledge; for example, implementing data type enhancements, ABAP proxies with attachments handling PDF and excel files – all of which are discussed in this chapter.

4.1 Trouble writing out PDF files using the File adapter?

Introduction

This article will discuss various options in SAP NetWeaver PI to create PDF file output. The reader will be introduced to a simple and easy to use API for creating PDF outputs which can be used in a module in a real time integration scenario.

Article

Creating a PDF output. What are the options?

1) Conversion Agent - Capable of literally taking in any kind of data and transforming out into any data, but comes with a license cost.
2) Using XSL-FOP and the Apache FOP in a XSLT mapping.
3) Use of Java proxies and executing scripts – may be complex to write.

Even though we have these options available, we recommend the use of a very simple to use free JAVA library called iText.

Let’s assume a typical scenario. There are a number of employee details that need to be written in PDF format. We need to utilize tables and include some simple formatting, such as colors and alignment.

The simplest design proposed is to have the mapping done, create the target XML with the employee details and then pass this to the Adapter (FILE/MAIL etc). Then use the MessageTransformBean and convert the XML to a FLAT format. Thereafter, insert the custom module with the code to create the PDF. The advantage here is that the input stream is in a flat string and it is very easy to do string manipulations.

If the MessageTransformBean is not used then the target XML in the module will be as input, then parse the content and build the PDF file using the iText library.

So to summarize, the message flow proposed is:

Target XML -> Adapter -> MessageTransformBean -> Our Custom Module

Below is a sample java class that is intended to help the reader understand the use of the iText library. The code can be used as a reference while building the EJB project for the module.

<u>Assumptions</u>

1) MessageTransformBean converts the target XML to a FLAT format. The field separator is comma and the end of each record can be identified by the string literal 'END'.
2) The iText library is downloaded and imported for the project.

e.g. Input format:

```
100,Anakin Skywalker,EAS,ConsultantEND200,Darth Vader,Java,Sr.
ProgrammerEND300,Obi-Wan Kenobi,MSTechmologies,Project
ManagerEND
```

There are 3 records and 4 fields per record.

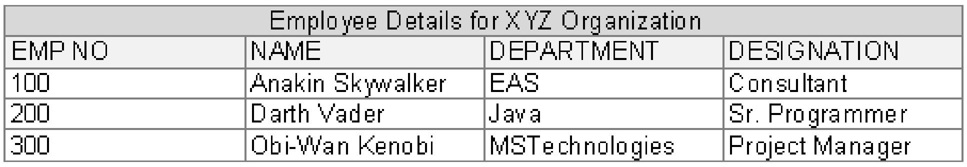
The output PDF will have a table which will include a Main Header, followed by the header for each column of the data, followed by the data itself.

```
/*
 This is a java class to create a PDF document out of a given string
 Created By Shabarish Vijayakumar
 */
import java.awt.Color;
import java.io.FileOutputStream;
import java.io.IOException;

import com.lowagie.text.Document;
import com.lowagie.text.DocumentException;
import com.lowagie.text.Element;
import com.lowagie.text.Paragraph;
import com.lowagie.text.pdf.PdfPCell;
import com.lowagie.text.pdf.PdfPTable;
import com.lowagie.text.pdf.PdfWriter;
public class createPDF {
   public static void main(String[] args) {

      //      Assume the below is the input file format
      String input =
         "100,Anakin Skywalker,EAS,ConsultantEND200,Darth
Vader,Java,Sr. ProgrammerEND300,Obi-Wan
Kenobi,MSTechnologies,Project ManagerEND";

      // creation of a document-object
      Document document = new Document();
      try {

         // create a writer
         PdfWriter.getInstance(
         // that listens to the document
         document,
         // and directs a PDF-stream to a file
         new FileOutputStream("output.pdf"));
         // open the document
         document.open();
         // add a table to the document
         PdfPTable table = new PdfPTable(4);
         PdfPCell cell =
            new PdfPCell(
               new Paragraph("Employee Details for XYZ Organization"));
         cell.setColspan(4);
         cell.setBackgroundColor(Color.red);
         cell.setHorizontalAlignment(Element.ALIGN_CENTER);
         table.addCell(cell);
```

```
//Set Header Text for the Table
cell = new PdfPCell(new Paragraph("EMP NO"));
cell.setBackgroundColor(Color.blue);
table.addCell(cell);

cell = new PdfPCell(new Paragraph("NAME"));
cell.setBackgroundColor(Color.blue);
table.addCell(cell);

cell = new PdfPCell(new Paragraph("DEPARTMENT"));
cell.setBackgroundColor(Color.blue);
table.addCell(cell);

cell = new PdfPCell(new Paragraph("DESIGNATION"));
cell.setBackgroundColor(Color.blue);
table.addCell(cell);

//Fill data to the table

String inputArray[] = input.split("END");

for (int i = 0; i < inputArray.length; i++) {

   String fieldValuesArray[] = inputArray[i].split(",");

   for (int j = 0; j < fieldValuesArray.length; j++) {

      table.addCell(fieldValuesArray[j].toString());
```

```
        }
      }

      document.add(table);
    } catch (DocumentException de) {
      System.err.println(de.getMessage());
    } catch (IOException ioe) {
      System.err.println(ioe.getMessage());
    }

    // close the document
    document.close();
–
  }
}
```

This is what the final output will look like (Figure 113):



Employee Details for XYZ Organization			
EMP NO	NAME	DEPARTMENT	DESIGNATION
100	Anakin Skywalker	EAS	Consultant
200	Darth Vader	Java	Sr. Programmer
300	Obi-Wan Kenobi	MSTechnologies	Project Manager

Figure 113

Note: When coding for the module, write out to an Outputstream rather than a FileOutputStream