Tapestry5HowToCreateADynamicPDF

Creating a Runtime Generated PDF

Here is a simple example of how to generate a PDF that gets streamed back to the user's browser. The main code was from an email exchange on the Tapestry User mailing list. This is similar to other examples that use StreamResponse for generating images and charts for the user.

Dependencies

This example uses the Lowagie IText library for generating the PDF.
It's hard to find a current version on maven repositories, so you will probably have to manually install it into your local maven repository. Please refer to the Maven documents on how to to manually add to your repository using mvn. You'll have to download the library from an online site. Google it. This is the same as with JFreeCharts. (Alternatively, you could just put the jar somewhere in your class path, if you are just testing things out.)

```
        <dependency>
                        <groupId>com.lowagie</groupId>
                <artifactId>itext</artifactId>
                <version>2.0.2</version>
        </dependency>
```

PDFWriterPage.html

First we have a very simple page. Notice that we've set the target to be a new window or tab. When the user click on the link, the PDF will be generated.

```
<html xmlns:t="http://tapestry.apache.org/schema/tapestry_5_0_0.xsd">
        <head>
        <title>PDF Writer test page</title>
        </head>
        <body>

        <t:form t:id="doReportsForm" action="reportsForm" target="_blank">

                <t:submit t:id="onSubmit" name="Run Report" value="Run Report" />

        </t:form>
        </body>
</html>
```

PdfWriterPage.java

Here is the Java class behind the simple page. It is here that you would supply the custom data for your PDF, or at least direct an object's output into the PDF generator. You also can (should) supply the filename here. This implementation does not require you to add the extention, it's added for you in a later class you write.

With a little work, you can use this for exporting the contents of a Grid component, for example. PDF is a hugely complicated format--the spec is bigger than a phone book. The Lowagie IText library makes it easy, and it's free. http://www.lowagie.com/iText/

```
import java.io.InputStream;

import myapp.model.services.PDFGenerator;
import myapp.model.services.util.PDFStreamResponse;

import org.apache.tapestry.StreamResponse;

public class PdfWriterPage {
        public StreamResponse onSubmit() {
                // Create PDF
                InputStream is = PDFGenerator.generatePDF("This is the content of a Dynamically Generated PDF");
                // Return response
                return new PDFStreamResponse(is,"MyDynamicSample");
        }
}
```

PDFGenerator.java

This is the class that takes care of your streams, and accepts your custom data. It returns a specific kind of stream for your page (PDFWriterPage) to use.

```
package myapp.model.services;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.InputStream;

import com.lowagie.text.Document;
import com.lowagie.text.DocumentException;
import com.lowagie.text.Paragraph;
import com.lowagie.text.pdf.PdfWriter;

public class PDFGenerator {

        public static InputStream generatePDF(String teststring) {
                // step 1: creation of a document-object
                Document document = new Document();

                ByteArrayOutputStream baos = new ByteArrayOutputStream();
                try {
                        // step 2:
                        // we create a writer that listens to the document
                        // and directs a PDF-stream to a file
                        PdfWriter writer = PdfWriter.getInstance(document, baos);
                        // step 3: we open the document
                        document.open();
                        // step 4: we add a paragraph to the document
                        document.add(new Paragraph(teststring));
                } catch (DocumentException de) {
                        System.err.println(de.getMessage());
                }
                // step 5: we close the document
                document.close();
                ByteArrayInputStream bais = new ByteArrayInputStream(baos.toByteArray());
                return bais;
        }
}
```

PDFStreamResponse.java

This is the type of object that the method in your page will return. Note the getContentType() method. If you are planning on returning something else, like a jpg, zip, mp3, etc., you will need to find the appropriate content type String for that. Otherwise, if it is wrong or blank, your browser may display it incorrectly or give you an error. These content types help trigger plugins associated with your browser.

Also notice the header being set in the prepareResponse() method. This is how your browser will know the default file name of the object you are streaming back. Since it also specifies "attachment", it will raise a file download box. Now if the user's browser has a plugin (i.e. Acrobat Reader) that intercepts the file type, it may simply open in a plugin and not display the file download box. You can read more about content-disposition here: http://www.ietf.org/rfc/rfc1806.txt

```
package myapp.model.services.util;

import java.io.IOException;
import java.io.InputStream;

import org.apache.tapestry.StreamResponse;
import org.apache.tapestry.services.Response;

public class PDFStreamResponse implements StreamResponse {
        private InputStream is;
        private String filename="default";

        public PDFStreamResponse(InputStream is, String... args) {
                this.is = is;
                if (args != null) {
                        this.filename = args[0];
                }
        }

        public String getContentType() {
                return "application/pdf";
        }

        public InputStream getStream() throws IOException {
                return is;
        }

        public void prepareResponse(Response arg0) {
                arg0.setHeader("Content-Disposition", "attachment; filename="
                                + filename + ".pdf");
        }
}
```

Note that the constructor above uses feature new to Java 1.5, the varargs feature.
It is included here simply as an introduction to it.

What it does in this case is allow you to create a PDFStreamResponse with or without supplying a filename. To do this in Java 1.4, we'd have to create a separate constructor that did not have a parameter for the filename.

So now, you can do this:

```
...
return new PDFStreamResponse(is, "DynamicSample");
...
```

(The filename will be DynamicSample.pdf) or this:

```
...
return new PDFStreamResponse(is);
...
```

(the filename will be default.pdf)

Resources

These resources should help you get up and running, so you can have more fun working with Tapestry:

A list of MIME Types:
http://www.fileformat.info/info/mimetype/index.htm

RFC for Content Disposition http://www.ietf.org/rfc/rfc1806.txt