SPML - Simple PDF Manipulation Language

Jayesh Kataria, jvk2104@columbia.edu
Dhivya Khrishnan, dsk2121@columbia.edu
Stefano Pacifico,sp2562@columbia.edu
Hye Seon Yi, hsy2105@columbia.edu

Programming Languages and Translator - Spring 2007
Prof. Stephen Edwards

February 7, 2007

Abstract

In this document we present the proposal for *SPML*, a simple, high-level programming language for PDF manipulation and analysis. We will discuss the main reasons behind this programming language and illustrate the key features we intend to implement. Also a brief syntax example will be provided. Keywords: Programming Languages, Compilers, PDF

1 Introduction

PDF (Portable Document Format) is a de-facto standard for electronic documents. Nowadays it is not unlikely, especially in the field of Computer Science, to have hundreds of documents stored into hard drives or other kind of supports. Many commercial products often do not provide the exact function needed to handle our electronic library, and accessing or manipulating PDF documents with common programming language is often hard and error-prone. Hence the need to provide a simple but effective programming language to rapidly build the tools one might desire to have in order to make the most out of his e-documents.

2 The idea

PDF is one of the most common and spread formats for e-documents: Adobe Systems Incorporated claims that there are more 200 million PDF documents available on the web; at present time Portable Document Format is under the process of ISO standardization; the query for "PDF" on Google search engine returns more than 1 billion results. Only considering these facts and figures, the utility of a programming language oriented to PDF documents handling appears clear.
More trivially, it happens often, especially in the scientific field, to have entire "virtual shelves" stuffed with papers, articles, books or chapters thereof. This situation implicitly yields the issue of interacting with such a large amount of documents. Quickly building ad-hoc scripts and applications for automated analysis, processing or generation of PDF documents is then mostly desirable.
The idea is to give programmers the tools to extract information from PDF documents, to create new ad-hoc PDF files based on the information extracted, to search into PDF files and so forth.

SPML compiler will be written entirely in JAVA. The compiler will convert the *SPML* source code into equivalent JAVA code, that eventually will be compiled by the Java compiler and executed through the Java Virtual Machine. This will ensure a high portability on various platforms even handheld and portable ones.
In conclusion, figures at hand, *SPML* appears not only as an interesting and useful development tool, but also as a very reasonable idea from a business perspective.

What exposed above justify, in the opinion of the authors, *SPML* as the language of choice.

3 Functionality

SPML will provide simple and convenient ways to create and manipulate PDF documents. Moreover, *SPML* will support functionalities linked to databases and text files so that programmers can use PDF with databases and text files flexibly and seamlessly. Many of these functionalities will be implemented using external libraries which are available for public, such as iText[1] and XPAAJ[2]. The main functionalities which *SPML* will be featuring follow in the paragraphs below.


```
[1] B. Lowagie, "iText Homepage".
http://www.lowagie.com/iText/
[2] Adobe, "Developing applicaions with XPAAJ".
http://www.adobe.com/devnet/livecycle/
       articles/developing_apps_with_xpaaj.html
```


3.1 PDF Generation

SPML will allow programmers to create PDF documents with selected settings, such as fonts, text colors, page size, and margin size. Paragraphs, list objects, images, and tables can be added to automate report creation and to generate contents dynamically.

3.2 PDF Documents Access and Manipulation

Programmers will be able to modify existing PDF documents. Examples are: concatenating multiple PDF files into one PDF file, combining a number of pages into one page, splitting a PDF file into different pages, and adding page number information. For instance, extracting the first pages of multiple PDF files and merging them together, creating thus a catalogue of the documents in a given directory or matching a certain criterion, is a task that could be easily accomplished with *SPML*.
Also, SPML can be used to perform searches at the word level. Given a PDF file, it is possible to extract words and determine the frequency of words from it. It will be also possible to find words in a PDF file that are not in another PDF file by simply ”subtracting” two PDF files.

3.3 Database Interfacing

It is intended to insert database accessing capabilities Using these database-related methods, programmers can generate a PDF document based on the contents of a database, or insert tabular informations from the PDF document to the database conveniently. In this implementation of *SPML*, it will be possible to connect to a single type of RDBMS only (e.g. MySQL). The user will be able to easily configure various parameters (like user name, password, data source name etc.) required to connect to a database and retrieve tables from it.

3.4 Text Files Support

The facilities for text files will allow programmers to utilize text files for editing PDF documents. For instance, text files can be converted into a PDF document or be used as to contain results of different processing applied to the PDF document in a straightforward way.

4 Language Features

The SPML lexicon consists of the following elements that can be used while constructing a program using the language.

4.1 Data Types

- *integer*: this data type will be identified by the *int* keyword. These will be used primarily for iterating in loops and indexing arrays.
- *strings*: this data type will be identified by using the *string* keyword and will be used to hold character data when the programmer needs to do so.
- *pdf*: this data type will be used for variables that hold references to PDF files. These variables can then be used to open existing PDF files or create new ones.
- *txt*: this data type will be used for variables that hold references to text files. They will be used to create new text files for manipulation or for storing intermediate data that could be retrieved later in the code.
- *database*: this type will be used to connect to an existing database and retrieve data from tables that could be converted to PDF files with formatting provided by the language.

See also the Syntax Example section for a pragmatic reference.

4.2 Programming Features

- *Looping Constructs*: looping constructs in the *SPML* language will be available as in the most common languages.
- *Flow Control*:

  ```
  if, then else
  ```

- *Operators*: arithmetic operators will be provided and illustrated in the sample code in the following section. We will be using the semicolon (;) punctuation mark as statement separator.
- *Functions*: function calling is intended to be a key feature in order to provide even more flexibility to the programmer. The code will start executing with the function called start().
- *Arrays*: array data structures will be designed and implemented.

5 Syntax Example

In this section we propose some code samples to illustrate the syntax of *SPML*.

A brief explenation of the code:
PDF variables and txt(i.e. text) variables are created. A database variable is also created. The code below exhibits the following functionalities:

a)An array of pdf variables can be created in the language.
b) Open existing PDF files and also create new PDF files.
c) A while construct that concatenates the PDF files in an iterative manner.
d) Changes made to a PDF file are saved.
e) Finding the difference of two PDF files
f) Combining two pages of a PDF into 1.
g) Highlighting a particular word in a PDF file.
h) Retrieving a particular page of a PDF file and storing it in a text file
i) Connect to a database, retrieve table contents and store it in a PDF file.

```
Start()
{
  /* Declaring a pdf variable */
  pdf file1;
  pdf file2;
  pdf file3,file4, file5;
```

```
/* Array of pdf variables */
pdf array[3];

/* Database variable */
database db;

/* String variable */
string s1;

/*Integer variable */
int i;

/* Text file variables*/
txt t1, t2, t3;

/*Loading a pre-existent PDF file into a variable*/
file1.open("C:\Sample1.pdf");
file2.open("C:\Sample2.pdf");

/* Creating a new PDF and setting title and author*/
file3.create("C:\Sample3.pdf", "Stefano");


/*Assignment*/
array[0]=file1;
array[1]=file2;
array[2]=file3;

i=0;

while(i<3) {
   file3=file3 + array[i];
   i=i+1;
}

/* Write the file*/
file3.save();

/*pdf file5 stores the difference of the words of the two files*/
file5.create("C:\Sample5.pdf", "Stefano");
file5=file1-file2;

/* contents of page 1 and 2 combined to page1*/
file2.combine(1,2);
```

```
 /* underline occurrence of zebra in file1*/
 file1.highlight(zebra);

 file1.save();
 file2.save();
 file5.save()
 /* retrieve page3 of file1 into text file t1 */
 t1 = file3.page(3);

 db.driver = <JdbcOdbcDriver>; /* Database Specifications*/
 db.dsn="a-dsn";
 db.username="user1";
 db.password= "u1";
 db.table="student";
 file4.getfromdatabase(db);

  /* Get table student from database and store it in file4 pdf file.
  file4.save();
}
```

6 Language Limitations and Future Work

SPML is not intended to have features for direct editing of PDF documents, like word insertion or replacement in pre-existent documents. A great path to follow for future development of *SPML* should be, thus, filling this gap, providing high level primitives for direct PDF manipulation and editing.
Also enhancing the database interface capabilities and adding more complex reporting primitives in *SPML* is highly desirable.