SPML

simple dfa manipulation language

Stefano Pacifico (sp2562@columbia.edu)

Jayesh Kataria (jvk2104@columbia.edu)

Dhivya Khrishnan (dsk2121@columbia.edu)

Hye Seon Yi (hsy2105@columbia.edu)

May 7, 2007

Content

1. Introduction: the proposal 5

1.1 The Idea 5

1.2 Functionality 6

PDF Generation 6

PDF Documents Access and Manipulation 6

Searches in a PDF file and directory 6

Text Files Support 7

2. Language Tutorial 7

2.1 First SPML program 7

2.2 Another program 8

2.3 Other Killer Applications of SPML 11

Delete a page 11

Creating two separate PDF files from a PDF file by separating them by even or odd page numbers 12

Get pages containing a phrase in a PDF 14

Get pages containing a phrase in all PDFs in a directory 16

Highlight every occurrence of a phrase in a PDF file 18

Reverse pages in a PDF file 18

Swap two pages in a PDF file 20

Convert a PDF file into a text file 22

Find which PDF files have the more number of pages 22

2.4 How to run SPML programs 23

For Unix / Linux platform 23

For Windows platform 23

3. Language Reference Manual 24

3.1 Lexical Conventions 24

Tokens 24

Comments 24

Identifiers 25

Keywords 25

Constants 25

Punctuations 26

3.2 Type Specifiers 26

Basic Types 26

Composite types 27
Void 28
3.3 Expressions 28
Arithmetic Expressions 28
Conditional Expressions / Relational Expressions 28
Logical Expressions 29
Expression Evaluation 29
3.4 Operators 29
Unary Operators 29
Multiplicative Operators 30
Additive Operators 30
Relational Operators 31
Equality Operators 32
Logical operators 32
Other operators 32
Operator Precedence 34
3.5 Statements 35
Variable Declarations 35
Assignment statements 35
PDF statements 36
Conditional statements 38
Iteration statements 38
Print statements 39
Jump statement 39
3.6 Functions 40
Keywords 40
Function Declarations 41
Function Definitions 41
Function Calls 42
Argument Evaluation 43
3.7 Scope 43
Lexical Scope 43
4. Project Plan 43
4.1 Processes Used 43
4.2 Programming Style Guide 44
Naming 44

Indentation and Spacing 44
Comments 44
4.3 Project Timeline 45
4.4 Roles and Responsibilities 45
4.5 Software Development Environment 46
4.6 Project log 46
5. Architectural Design 47
Front End 49
Tree Walker 49
Back End 53
Runtime Libraries 54
6. Test Plan 55
7. Lessons Learned 56
8. Appendix 59
Front End 59
lexerparser.g 59
Tree Walker 67
CompilerException.java 67
Env.java 68
Iden.java 72
PDFItem.java 73
Prototype.java 74
SpmlAST.java 77
SpmlCodeGen.g 78
SpmlOut.java 108
SpmlWalker.g 109
SymbolTable.java 138
Type.java 139
TypeArray.java 144
TypeInt.java 144
TypePDF.java 144
TypeString.java 145
TypeVoid.java 145
Variable.java 146
Code Generation 149
CodeGen.java 149

SPMLLibrary.java 170
Run SPML source code 183
Compile.sh 183
Main.java 184
SPML.bat 186
Test 186
test-walker.sh 186
Testing Lexer / Parser / Walker 187
Testing programs 187
Testing types 189
Testing if 190
Testing while 190
Testing print 191
Testing in 192
Testing functions 192
Tesing Code Generation 194

1. Introduction: the proposal

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.

1.1 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.

1.2 Functionality

SPML will provide simple and convenient ways to create and manipulate PDF documents. However, the focus of SPML functionalities is on manipulation over creation of PDF files since creating PDF files can be easily accomplished by installing freeware like PDF ReDirect[1] and cutePDF[2] Writer. Moreover, SPML will support functionalities linked to text files so that programmers can use PDF with text files flexibly and seamlessly. Many of these functionalities will be implemented using external libraries which are available for public, such as iText[3]and XPAAJ[4]. The main functionalities which SPML will be featuring follow in the paragraphs below.

PDF Generation

SPML will allow programmers to create PDF documents from existing PDF files. Programmers can set the filename for a new PDF file and the context by extracting information from existing PDF files.

PDF Documents Access and Manipulation

Programmers will be able to modify existing PDF documents. Examples are: concatenating multiple PDF files into one PDF file, splitting a PDF file into different pages and highlighting a phrase in one PDF file. 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.

Searches in a PDF file and directory

---

[1] Download.com, “PDF ReDirect”. http://www.download.com/PDF-ReDirect/3000-6675-10255233.html

[2] CutePDF, “CutePDF ideas for PDF”. http://www.cutepdf.com/

[3] B. Lowagie, "iText Homepage". http://www.lowagie.com/iText/

[4] Adobe, "Developing applicaions with XPAAJ".

http://www.adobe.com/devnet/livecycle/articles/developing_apps_with_xpaaj.html

SPML can be used to perform searches at the word level. Given a PDF file, it is possible to find page numbers where a word or phrase is used. Also, SPML will allow programmers to find one specific or all PDF files in a directory given with a path.

Text Files Support

The facilities for text files will allow programmers to utilize text files to index words and phrases. For instance, a PDF document can be converted into a text file.

2. Language Tutorial

SPML provides means and ways to manipulate PDF documents. This tutorial will help programmers learn SPML as easily and quickly as possible.

2.1 First SPML program

The first SPML program combines two different PDF files and save them into a new PDF file.

```
/*                                                                    1
 * The program create a new PDF file called c.pdf
 * by combining the first page of a.pdf and
 * the first page of b.pdf.
 */                                                                   5
start()                                                               6
{                                                                     7
   /* open a.pdf */
   pdf p1;                                                            9
   p1 = "a.pdf";                                                      10

   /* open b.pdf */
   pdf p2;                                                            13
   p2 = "b.pdf";                                                      14
```

```
    /* combine those two PDF files and
     set the filename to c.pdf and the author to user */
    pdf combined;                                                    18
    combined = create "c.pdf";                                       19
    combined = p1 + p2;                                              20
}                                                                    21
```

Line 1 to 5 of the code show how to comment SPML programs. Any sequence of characters enclosed by “/*” and “*/” are considered as comments in SPML. However, comments cannot be nested. Comments are used for providing information to programmers but does not affect the execution of a program since they are ignored.

SPML programs consist of functions which contain statements. Line 6 has a function definition for start(). start() is a special function where a program is called for execution. Thus, SPML programs without start() cannot be executed. start() cannot take an input or return any output. Except start(), programmers can define a user defined function with a name (as long as it follows a naming rule for identifiers), arguments and return type of their choice. The body of start() is from Line 7 to 21, which contains the statements of the program. Each statement is separated by a semicolon.

Line 9 and 10 shows how to open a PDF file. A pdf variable should be declared at first and then assign the filename to it since SPML does not allow a variable to be declared and initialized at the same time. After this statement, a pdf variable called p1 will refer to “a.pdf” file until it is referred to something else. If “a.pdf” does not exist, a runtime error will be thrown. Line 13 and 14 open “b.pdf” and refer it using the pdf variable p2.

Now is the time to combine these two PDF documents. This is shown lines 18 to 20. At line 19, a pdf variable referring to a new PDF file called c.pdf in the current directory is created. However, the actual PDF file is not created in the directory yet. The create statement allows programmers to create a new file if there is no file with the same name, or to overwrite to the existed file if a file with the same name already exists. At line 21, two PDF files (a.pdf and b.pdf) are combined using the addition operator (+). This program demonstrates well that the hard task of manipulating PDF files can be done easily using SPML.

2.2 Another program

This program creates a “sitemap” of PDF files in a directory. The “sitemap” will be a PDF file which consists of the first page of all PDF files in a directory.

```
  /*                                                                  1
* Create a “sitemap” PDF file which consists of
* the first page of all PDF files in a directory
*/                                                                  4

  /* pdfs_in_directory() function declaration */
  pdf[] pdfs_in_directory (string dir);                             7

  /* execution point */
  start()                                                           10
  {                                                                 11
   /* create an array of all PDF files
        in the current directory */
     pdf files[10];                                                 14
     files = pdfs_in_directory(“.”);                                15

 /* find the length of the array of PDF files */
     int len;                                                       18
     len = length files;                                            19

     /* create a sitemap.pdf file */
     pdf sitemap;                                                   22
     sitemap = create “sitemap.pdf”;                                23

     /* while looping through the array of PDF files,
      add each first page of PDF files to sitemap */
     int idx;                                                       27
     idx = 0;                                                       28
     while (idx < len)                                              29
     {                                                              30
        /* extract one page of a PDF file */
        pdf tmp;                                                    32
```

```
        tmp = extractpage files[idx] 1;                          33
        /* add the page to sitemap */
        sitemap = sitemap + tmp;                                 35
        idx = idx + 1;                                           36
     }                                                           37
  }                                                              38

  /*
* Return an array of PDF files in a directory
* whose path is passed as an argument
*/
  pdf[] pdfs_in_directory (string dir)                           44
  {                                                              45
     pdf files[10];                                              46
     files = pdf in dir;                                         47
     return files;                                               48
  }                                                              49
```

Lines 1 to 4 are comments about the program. Line 7 shows how to declare a function before its usage and definition. The function is declared as a function called `pdfs_in_directory` which takes a `string` argument and returns an array of `pdf`. All function declaration except start() should come before start(), which does not need a declaration.

`start()` is an execution point of SPML as explained in 2.1. The body of start() is from line 11 to line 34. Line 14 declares an array of pdf. An array can store a variable number of items with the same type. Line 15 retrieves all PDF files in the current directory using `pdfs_in_directory()` function which was declared before start() and defined after start(). Line 18 and 19 are an example of the length operator. The length operator is used to find the length of an array or the total number of pages of a PDF file. Line 22 and 23 create a PDF file called `sitemap.pdf`.

Lines 27 to 33 illustrates how to use a loop statement. while is the only loop statement provided in SPML which iterates the body of the loop as long as the condition (enclosed in parentheses right after the while keyword) is true. In SPML, 0 denotes false and any number except 0 denotes true. Thus, the condition of the while, idx < len, will be evaluated true as long as idx is smaller than len. The body of while is from line 30 to line 36. Line 32 and 33 extract one page of a PDF file. Line 35 adds

this extracted page to sitemap pdf variable using "+" operator.

Lines 43 to 48 define the `pdfs_in_directory` function declared before start(). The function should have the same return type and argument type as the declation, but the name of the argument can be changed. Line 46 finds all PDF files in a directory using the in operator. There are four use cases with the in operator. The first use case is finding all PDF files in a directory as this example. The second use case is searching a word in a PDF document. The third use case is finding whether one string is a substring of the other. The last use case is when in operator is used in the if statement, which returns whether a PDF file exists in a directory. The subpart, "Other operators" under "3.4 Operators" in language reference manual explains this in more detail. `pdfs_in_directory` () returns the array of PDF files to its caller function.

2.3 Other Killer Applications of SPML

Delete a page

```
/* To delete a page 5 from the src file */
pdf returnPage(pdf src,int num);

start()
{
	pdf src ;
	int i;
	i=1;
	src = "C:\\CN-Paper.pdf";
	int delPage;
	delPage = 5;
	int numOfPages;
	numOfPages = length src;
	if(delPage > numOfPages)
	{
		print "The page " +delPage + " does not exist";
	}
	else
	{
```

```
                pdf dest;
                i =1;
                dest = returnPage(src,i);
                i= i + 1;
                while(i<=numOfPages)
                {

                        if(i==delPage)
                        {
                                i=i+1;
                                continue;
                        }
                        dest = dest+ returnPage(src,i);
                        i = i+1;
                }

        src = dest;   /* this modifies the source file */

        }

}

pdf returnPage(pdf src,int num)
{
        pdf temp;
        temp = extractpage src num;
        return temp;

}
```

Creating two separate PDF files from a PDF file by separating them by even or odd page numbers

```
int checkeven(int a);

start(){
```

```
pdf evenPdf;
pdf oddPdf;
pdf src;
pdf temp;

src = "F:\\PLT-prj files\\Testing\\pdfs\\CN-Paper.pdf";
evenPdf = create "F:\\PLT-prj files\\Testing\\pdfs\\evenCN.pdf";
oddPdf = create "F:\\PLT-prj files\\Testing\\pdfs\\oddCN.pdf";

int pages;
pages = length src;

int count;
count =1;
int j;
j=1;
while(j<=pages)
{

 if(checkeven(j)==1)   /* it is even*/
 {
   if(count==1)
        {
      evenPdf = extractpage src j;
      count=0;

   }
   else
   {
         temp = extractpage src j;
         evenPdf = evenPdf + temp;
   }
 }

 else   /* page is odd */
```

```
    {
      if(count==1)
           {
              oddPdf = extractpage src j;
      }
      else
      {
       temp = extractpage src j;
            oddPdf = oddPdf + temp;
      }
    }
  j = j+1;
   }
}

int checkeven(int a){
 int b,c;
 b = a/2;
 int result;
 result = 5;
 c = b*2;
 if (c == a)
 {
  result=1;
 }
 else
 {
  result = 0;
 }

 return result;

}
```

Get pages containing a phrase in a PDF

```
start()
{
pdf p;
pdf finalp;

p = "F:\\PLT-prj files\\Testing\\pdfs\\ANTLR.pdf";
finalp = create "F:\\PLT-prj files\\Testing\\Result.pdf";
int count;
count =0;
int arr[100];
arr = "Grammar" in p;
int a;
a = length arr;
int j;
j=0;
while(j<a)
{
        pdf temp;
    temp = extractpage p arr[j];
    if(count==0)
    {
            finalp = extractpage p arr[j];
    }

    else
    {

            finalp = finalp + temp;
    }

    count=1;
    j = j+1;
}

}
```

Get pages containing a phrase in all PDFs in a directory

```
string getWord();
start()
{
	pdf finalPdf;
	string path;
	path = "F:\\PLT-prj files\\Testing\\getwordpdfs.pdf";
	finalPdf = create path;
	pdf dirPdfs[40];
	dirPdfs = pdf in "F:\\PLT-prj files\\Testing\\pdfs" ;
	int lenDir;
	lenDir = length dirPdfs;
	int count;
	count =0;
	print "Lendir is " + lenDir;
	int ix;
	ix = 0;
	pdf temp;
	pdf temp2;

	if(lenDir!=0)
	{

		while(ix<lenDir)
		{

			temp = dirPdfs[ix];

			int j;
			j = 0;
			int arr[100];
			arr = getWord() in temp;
			int pgnumbers;
			pgnumbers = length arr;
			print "pgnums is " + pgnumbers;
```

```
                while(j < pgnumbers)
                {

                        print ix + " ";
                        print j;
                        temp2   = extractpage temp arr[j];


                        if(count==0)
                        {
                            finalPdf = extractpage temp arr[j];
                            count=1;
                        }
                        else
                        {
                            finalPdf = finalPdf + temp2;
                        }

                        j = j+1;
                }


                ix = ix+1;
            }

    }

    else
    {
        print "Sorry, no Pdfs in the directory - " + path ;
    }


}
```

```
string getWord()
{
      return "COMS";
}
```

Highlight every occurrence of a phrase in a PDF file

```
/* Highlight every occurence of the word */

string getPath();
string getWord();
start()
{
pdf p;
string path,word;
path = getPath();
word = getWord();

/* highlight the path */
p = path;
highlight p word;
}

string getPath()
{
return "F:\\PLT-prj files\\Testing\\pdfs\\CN-Paper.pdf";
}

string getWord()
{
string w;
w = "system";
return w;
}
```

Reverse pages in a PDF file

```
/* program showing recursion   - reverses the pages*/

void reverse(pdf src,pdf dest,int x,int count);
start()
{
	string path;
	path = "F:\\PLT-prj files\\Testing\\pdfs\\CN-Paper.pdf";
	pdf src;
	src = path;

	pdf dest;
	dest =   create "F:\\PLT-prj files\\Testing\\Reversed.pdf";

	int num;
	num = length src;
	reverse(src,dest,num,1);

}

void reverse(pdf src,pdf dest,int num,int count)
{

	if(num==0)
	{
		return;
	}

	if(count==1)
	{
		dest = extractpage src num;
	}
	else
	{
		pdf temp;
		temp = extractpage src num;
```

```
            dest = dest + temp;
        }

        reverse(src,dest,num-1,0);
    }
```

Swap two pages in a PDF file

```
    pdf returnPage(pdf src,int num);
    start()
    {
        pdf src ;
        src = "C:\\CN-Paper.pdf";
        int swap1,swap2;
        int numOfPages;

        swap1 = 5;
        swap2 = 6;
        numOfPages = length src;

        pdf dest1;
        int count,j;
        count=1;
        int stop;
        stop = swap1-1;
        j=1;
        dest1 = returnPage(src,1);

        while(j<=stop)
        {
            if(count==1)
            {
                dest1 = returnPage(src,j);
                count=0;
            }
            else
```

```
		{
			dest1 = dest1 + returnPage(src,j);

		}

		j = j+1;
	}

	dest1 = dest1 + returnPage(src,swap2)+returnPage(src,swap1);

	j=swap2+1;
	stop=numOfPages;
	while(j<=stop)
	{
		if(count==1)
		{
			dest1 = returnPage(src,j);
			count=0;
		}
		else
		{
			dest1 = dest1 + returnPage(src,j);

		}

		j = j+1;
	}

	src= dest1;

}

pdf returnPage(pdf src,int num)
{
	pdf temp;
	temp = extractpage src num;
```

```
        return temp;
    }
```

Convert a PDF file into a text file

```
    start()
    {
        string path;
        path = "C:\\CN-Paper.pdf";
        pdf src;
        src = path;
        totextfile src "C:\\CN-Paper.doc";
    }
```

Find which PDF files have the more number of pages

```
    /* to find which pdf is bigger in terms of number of pages */
    start()
    {
        string path1,path2;
        path1 = "F:\\PLT-prj files\\Testing\\pdfs\\CN-Paper.pdf";
        path2 = "F:\\PLT-prj files\\Testing\\pdfs\\highlight.pdf";

        int num1;
        int num2;

        pdf p1,p2;
        p1 = path1;
        p2=path2;

        num1=   length p1;
        num2 = length p2;

        if(num1>num2)
        {
                print "Pdf in "+path1+" has more number of pages :)";
```

```
        }

        else
        {
                if(num1<num2)
                {
                        print "Pdf in "+path2+" has more number of pages :)";
                }
                else
                {
                        if(num1==num2)
                        {
                                print "Both pdfs have the same number of pages";
                        }
                }
        }
}
```

2.4 How to run SPML programs

For Unix / Linux platform

To run a SPML source code in Unix/Linux, compile.sh is required. compile.sh is a shell program which takes a SPML source code as an input and run it. The path information in the shell program should be adjusted according to the programmer's environment. The below is an example of using compile.sh.

```
$ compile.sh sample.spml
```

For Windows platform

To run a SPML source code in Windows, SPML.bat is required. These are the two steps to run SPML programs.

- SPML.bat has been developed to compile and run programs written in SPML

in Windows. This batch file contains the classpath of the jar files that are used to run SPML programs namely, iText, XPAAJ and Antlr. The path should be modified to reflect the machine it is run on and should also contain the path to the folder where the programs are run. This batch file should be placed directly above the "spml" package in the folder hierarchy.

- As an example, the command to run the batch file to compile a sample file sample.spml would be

```
C:\SPML E:\\final_plt\\FinalDemo\\sample.spml sample.java
   sample
```

3. Language Reference Manual

3.1 Lexical Conventions

Tokens

Tokens are divided in the following classes:

Identifiers
Keywords
Constants
String Literals
Operators
Other Separators

Blanks, horizontal and vertical tabs, newlines, formfeeds and comments are considered as “white space”. They have no semantic meaning, and are used only as token separators. The input stream is divided into tokens where each token is a group of valid characters that could constitute a token.

Comments

Comments are defined as a stream of characters that begin with the sequence **/*** and

end with the sequence */ . Comments cannot be nested and cannot occur inside a string or character literals.

Identifiers

An identifier is a sequence of letters and digits. The first letter must be a character between [A-Z] or [a-z]. The remaining characters can be from [A-Z] or [a-z] or digits from [0-9] or the underscore (_) character.

Keywords

The following identifiers have been reserved as keywords and cannot be used otherwise

break	**continue**	**create**	**else**	**extractpage**
highlight	**if**	**in**	**int**	**pdf**
print	**return**	**start**	**string**	**totextfile**
void	**while**			

Constants

The different kinds of constants supported by SPML are :

- **Integer Constants**

SPML only supports integer constants to the base 10 i.e. decimal integers. Constants in the octal or hexadecimal number system are not supported. All integer constants are signed constants. Unsigned constants are not supported by SPML.

- **String Constants**

String constants, also called string literals, are sequences of characters enclosed in double quotes for example **"**`abc\n`**"**. The escape sequence '\n' denotes a new line and it can be used in any string constants. Below is a table of escape sequences in SPML.

`\b`	backspace
`\t`	tab
`\n`	newline
`\r`	carriage return
`\"`	double quote
`\'`	single quote
`\\`	backslash

Punctuations

These are the punctuation symbols in SPML.

;	statement end
,	argument separator
“ ”	string constant
[ ]	range or array index delimiter
{ }	function body or block delimiter
( )	function argument list

3.2 Type Specifiers

Basic Types

In SPML, there are three basic types available.

- **int**

int is the primitive type for representing integer numbers. The int type is compatible with the Java int type, which means each int is 4 byte-long.

- **string**

string is the primitive type for representing character strings. The string type is compatible with the Java String type, which can hold a set of characters in variable length.

- **pdf**

pdf is the primitive type representing PDF documents. Thus, the pdf type is used as a reference type of PDF documents.

Composite types

Composite types are composed of the basic types. array is a composite type which can store or reference a variable length of data items with one same basic type. The following syntax is used to declare an array.

```
<basic type> <array name> [length];
```

The basic type can be int, string, and pdf. Users can set a random name for a variable name as long as the name follows the rules for identifiers. An array name is used to refer to the elements of an array. Brackets are used to specify the length of an array. The length of an array is required in declaration. An array name can be used as a holder for dynamic array creation, for another array, or for returning an array type for a function. After the length of an array is declared, each element of the array can be referred or manipulated with an array index. The starting index is always 0. For example, the following statement defines an array of three integers.

```
int arrayInt[3];
```

The elements of arrayInt are referred as arrayInt[0], arrayInt[1], and arrayInt[2]. The array name can refer to different set of elements when the elements type is the same.

```
int arrayA[1];[1]
int arrayB[2];

arrayA [0] = 0;
arrayB [0] = 1;
arrayB [1] = 2;
```

```
arrayA = arrayB;[2]
```

In the example above, line [1] declares arrayA to point to an integer array of length 1. At line [2], arrayA points an integer array of length 2. This assignment is permitted since the type of elements is both int. After line [2], the memory where arrayA has pointed (an integer array of length 1) is discarded since there is no reference to it.

Void

void is a type to symbolize non-existence of a return value in function declarations and definitions. Thus, if a return type is void, there is no return value of the function.

3.3 Expressions

Different types of expression are supported in SPML.

Arithmetic Expressions

Arithmetic Expression include arithmetic operators. An example is `a+b-c*d/f`. Additionally, the **+** operator can be applied on PDF documents. It is used to combine two PDF documents into one. The following example illustrates the combination of PDF documents referenced by p1 and p2 into one PDF document referenced by p3.

```
pdf p1, p2, p3;
…
p3 = p1 + p2;
```

Conditional Expressions / Relational Expressions

Conditional Expressions are those involving comparison operators such as <, <=, >, >=, == and !=.

They can be used inside of if and while loops in SPML in order to test conditions. The associativity of these operators is from left to right. All of these operators are binary operators that work on two operands. Examples are a<b, b>=d.

Logical Expressions

Logical Expressions can only take the values for true or false. In SPML, 0 denotes false while all the other integer values represent true. The logical operators supported by the language are &&, || and !. && represents the AND logic, || represents the OR logic and ! is the NOT operator. These operators can be used with relational expressions. An example is

```
if(a<b && b>c)
```

If a > b > c, the condition in if is evalued false. The same can be applied to the while loop as well. ! has the highest precedence, followed by && and then by ||.

Expression Evaluation

Expressions are all evaluated left to right. In case of logical expressions, if the evaluation of the first n elements of the expression is sufficient to evaluate the whole expression itself, the n+1th element will not be evaluated.

3.4 Operators

Unary Operators

Unary operators affect the expressions on their right.

- **Minus sign: ~** *expression*

In SPML, the minus sign operator is ~ not -. ~ represents the negative value of *expression*. *expression* must be the int type.

- **Logical negation: !** *expression*

The result of the logical negation operator *!* is the constant 1 of the int type if *expression* is evaluated as 0; constant 0 of the int type if *expression* value is otherwise. *Expression* must be the int type.

- **Variable length: length** *expression*

The result of the length operator on an array is an int value of the number of elements contained in *expression* if *expression* is the array type.

It returns the number of pages contained in the PDF document referred by *expression*, if *expression* is the pdf type.

Multiplicative Operators

Multiplicative operators associate expressions left-to-right.

- **Multiplication:** *expression1* ***** *expression2*

The multiplication operator ***** returns an int value of the product of the values of *expression1* and *expression2*. *expression1* and *expression2* must be the int type.

- **Division:** *expression1* **/** *expression2*

The division operator **/** returns the integer quotient of the int type between the values of *expression1* and *expression2*. *expression1* and *expression2* must be the int type. Division by 0 is not allowed and will cause a runtime error.

Additive Operators

Additive operators associate expressions left-to-right.

- **Addition:** *expression1* **+** *expression2*

The addition operator **+** returns the sum of the values of *expression1* and *expression2* if both *expression1* and *expression2* are the int type. It returns an object of the pdf type being the concatenation of the PDF files referred by *expression1* and *expression2* if both *expression1* and *expression2* are the pdf type. Any PDF file referred by *expression1* or *expression2* not being accessible at runtime will cause a runtime error.

If one of *expression1* and *expression2* is the string type and the other one is the

int type**,** the expression of the int type is converted into a string to be added to the other expression**.** In the example below, s2 will contain a string “string1” and s3 will contain a string “1” after running the code.

```
int i;
i = 1;

String s1;
s1 = “string”;

String s2;
s2 = s1 + i;

String s3;
s3 = “” + i;
```

If both of *expression1* and *expression2* are the string, it returns a string which is their concatenation. No other type combination is allowed.

- **Subtraction:** *expression1* **–** *expression2*

The subtraction operator **–** returns an int value of the difference between the values of *expression1* and *expression2*. *expression1* and *expression2* must be the int type**.**

Relational Operators

- **Greater than:** *expression1* **>** *expression2*
- **Less than:** *expression1* **<** *expression2*
- **Greater than or equal:** *expression1* **>=** *expression2*
- **Less than or equal:** *expression1* **<=** *expression2*

The greater than **>**, greater than or equal **>=**, less than **<**, less than or equal **<=** operators return constant 1 or 0 of the int type if the relation between *expression1* and *expression2* is true or false respectively, in the numerical order and both *expression* and *expression2* are the int type**.** They return constant 1 or 0 of the int

type if the relation between *expression1* and *expression2* is true or false respectively in the alphabetical order and both *expression1* and *expression2* are the string type**.** No other type combination is allowed.

Equality Operators

- **Equality:** *expression1* **==** *expression2*
- **Inequality:** *expression1* != *expression2*

The equality **==** and inequality **!=** operators return constant 1 or 0 of the int type if *expression1* and *expression2* have the same value or not respectively and both are the int type**.** They return constant 1 or 0 of the int type if *expression1* and *expression2* refer to the same PDF file or not respectively, and both *expression1* and *expression2* are the pdf type. They return constant 1 or 0 of the int type if *expression1* and *expression2* are the same string or not respectively, and both *expression1* and *expression2* are the string type.

Logical operators

- **Logical AND:** *expression1* **&&** *expression2*

The logical AND operator **&&** returns constant 1 of the int type if *expression1* and *expression2* are evaluated to non zero for both. It returns 0 otherwise.

- **Logical OR:** *expression1* || *expression2*

The logical OR operator || returns constant 0 of the int type if *expression1* and *expression2* are evaluated to 0 for both. It returns 1 otherwise.

Other operators

- **In:** *expression1* **in** *expression2*

The element of operator in returns an array of the pdf type elements referring to all the PDF documents contained in the directory as for the path indicated in *expression2*; in this case *expression1* must be the pdf type keyword and

expression2 must be one of these: a string literal, a string variable, an element of a string array or a function call returning a string value. The following is an example returning all PDF files under a directory called "`dir`" into `group` pdf array.

```
pdf group[10];
   group = pdf in "dir";
```

The in operator returns an array of the int type elements containing the page numbers where *expression1* occurs in *expression2*; in this case *expression1* must be one of these: a string literal, a string variable, an element of a string array or a function call returning a string value whereas *expression2* must be one of these: a pdf variable, an element of a pdf array or a function call returning a pdf reference. If The PDF file referred by *expression2* cannot be opened, a runtime error will be thrown. The following is an example returning all line numbers which contain a word "`the`" in a PDF file called "`ex.pdf`".

```
pdf p;
   p = "ex.pdf";
   int a[10];
a = "the" in p;
```

The in operator returns constant 1 of the int type if *expression1* is a substring of *expression2* and 0 if otherwise. In this case both *expression1* and *expression2* must be one of these: a string literal, a string variable, an element of a string array or a function call returning a string value. The value of `a` will be `1` after the execution.

```
int a;
a = "1" in "12345";
```

The in operator returns constant 1 of the int type if a pdf document referred by *expression1* exists under a directory represented by *expression2*; in this case *expression1* must be one of these: a pdf variable, an element of a pdf array or a function call returning a pdf reference and *expression2* must be one of these: a string literal, a string variable, an element of a string array or a function call returning a string value. The following example to check if a PDF file called "`ex.pdf`" is residing in a directory called "`dir`".

```
pdf p;
p = "ex.pdf";
if (p in "dir"){
   ...
}
```

No other combinations of types are allowed.

- **Extract page from PDF: extractpage** *expression1 expression2*

The extract a page from PDF operator extractpage returns an object of the pdf type referring to a new PDF document made of the page number *expression2* extracted from the PDF file referred by *expression1*. *expression1* must be one of these: a pdf variable, an element of a pdf array or a function call returning a pdf reference; *expression2* must be one of these: a integer literal, an integer variable, an element of a int array or a function call returning an integer value.

Operator Precedence

The following operator precedence list for SPML from the highest to the lowest. The operators with the same precedence are placed in the same line.

[]
~ ! length
*** /**
+ -
in
< > <= >= == !=
&& ||
=
,

All binary operators associate left-to-right whereas the unary operators associate right-to-left.

3.5 Statements

Below are the descriptions of the statements in SPML.

Variable Declarations

These statements are used to declare the variables to be used by the program. The declarations are terminated with a semicolon and declarations can be separated by the comma operator. Below are some examples of variable declarations.

```
int p1, p2, p3;
int p1;
```

Array declarations are also supported. One example is

```
int a[10];
```

Declarations of array and variables can be given in one statement. However, value assignments should be performed in a separated statement.

```
int a, b[5];[1] /* allowed */
int c = 1;[2] /* error */
```

The declaration [1] consists of a variable of the int type and an array of five int. These two can be given in any order. The other two possible variable declarations are the ones for pdf and string types. The statement [2] is not allowed in SPML since declaration and assignment should be separated.

Assignment statements

Assignment statements enable the programmer to define or redefine a symbol. The right side of an assignment statement can be an arithmetic expression. The syntax is the following:

```
Variable = expression;
```

This is an example where the value 5 is assigned to the variable x .

```
int x;
x = 5;
```

The example below is makes the pdf variable p refer to the existing a PDF file called “file.pdf”.

```
pdf p;
p = “file.pdf”;
```

If file.pdf does not exist, a runtime error will be thrown. In SPML, the type of the operands on both sides of the assignment operator **=** must be the same. For example:

```
p1 = p2 + p3;
```

In this case, type of the left side (p1) and the type of the right side (p2 and p3) should be one of the int, string, or pdf types. Thus, type conversion is not allowed in SPML.

PDF statements

The following statements involving pdf type variables are supported.

- **highlight statement**

  highlight statement underlines every occurrence of the word in a PDF document. The first argument must be one of these to reference a PDF document: a pdf variable, an element of a pdf array or a function call returning a pdf reference. The second argument must be one of these to represent a word or phrase to be highlighted: a string literal, a string variable, an element of a string array or a function call returning a string value. An example of highlight statement is:

  ```
  highlight pdf_variable “hightlighted_word”;
  ```

This underlines every hightlighted_word in the PDF document referred by pdf_variable.

- **extractpage statement**

This statement is used to extract a page of a PDF document file into a PDF file.

```
pdf pdf_variable;
pdf_variable = “C:\p1.pdf”;
pdf p;
p = extractpage pdf_variable 1;
```

The statement extracts page 1 of p1.pdf which is referred by pdf_variable into the pdf variable p.

- **create statement**

The create statement is used to denote that a new PDF document file should be created even if there is a file with the same name. The first argument must be one of these to refer to a file object which will be created: a pdf variable, an element of a pdf array or a function call returning a pdf reference. The second argument must be one of these to represents the path for the file object: a string literal, a string variable, an element of a string array or a function call returning a string value. Since create statement does not specify the context of a PDF file, another content manipulation operation is required to create a PDF file on disk.

```
pdf pdf1, pdf2;
pdf1 = “file.pdf”;
pdf2 = create “C:\test.pdf”;[1]
pdf2 = extractpage pdf1 1;[2]
```

The statement[1] creates a reference to a new PDF document named test.pdf. However, the actual file has not been created yet. The test.pdf will be created on disk after the statement[2], which setting Page 1 to test.pdf.

- **totextfile statement**

This statement produces a text file version of a PDF document. The first argument must be one of these to refers to a PDF document: a pdf variable, an element of a pdf array or a function call returning a pdf reference. The second argument must be one of these to represent a path for a text file created: a string literal, a string variable, an element of a string array or a function call returning a string value. Below is an example:

```
pdf file1;
totextfile file1 "/path/to/textfile";
```

This example creates a text file "path/to/textfile" which contains the contents of a PDF document referred by file1.

Conditional statements

Conditional statements define control flow based on a condition tested. SPML supports the **if..else** conditional construct. The syntax is

```
if(condition)
{
   (statement)*
}

else
{
   (statement)*
}
```

The condition must be an expression with an int value. If the expression evaluates to a value different from 0, the if block is executed. Otherwise, the else block is executed. Also, each block should be enclosed by curly brackets.

Iteration statements

Iteration statements are used to execute a block of statements more than once

instead of replicating code. SPML supports the **while** iteration statement, the syntax of which is the same as C.

```
while(condition)
{
   (statement)*
}
```

If the condition evaluates to a value different from 0 then the while block is executed. Otherwise, the while block is not executed. condition must be an expression with an int value. Also, the while block should be enclosed by curly brakets.

Print statements

Print statements are used to display the value of an expression on the screen. Print statements take one of these for printing: a string literal, a string literals connected with '+', a string variable, an element of a string array or a function call returning a string value. Thus, the syntax is:

```
print expression;
```

The following example prints '1' on the screen.

```
int a;
a = 1;
string s;
s = "a = " + a + "\n";

print s;
```

Jump statement

Jump statements are used to jump to a different set of statements altogether. They affect the control flow immediately without any condition check as in iterative or conditional statements.

- **return**

A return statement is used in functions to return a single value or void. A return statement can be used any functions except start function.

- **continue**

A continue statement is used to continue with the next iteration in a while loop construct. Thus, a continue statement can be used only in a while loop but anywhere else.

- **break**

A break statement is used to break out of a while loop immediately. Thus, a break statement can be used only in a while loop but anywhere else.

3.6 Functions

There are two types of functions: language defined functions and user defined functions. Users are able to define functions.

Keywords

These are three keywords related to functions.

- **return**

A function returns to its caller by the return statement. All user defined functions need to have at least one return statement. However, return statement can be omitted in the start() function.

- **start**

start() is a language defined function which is the execution point in SPML programs. In other words, the program cannot be executed without a start function.

- **void**

  void is placed to symbolize non-existence in function declaration. If a return type is void, there is no return value of the function. If void is used as an argument, there is no argument needed to call the function.

Function Declarations

To define and use a user-defined function, the function should be declared with a function prototype before start function. A function declaration has three sub parts namely: a return type, a function name, and a list of arguments.

```
<return type> <function name> (argument list);
```

A return type can be any of types available in SPML, which are int, string, pdf, array composite type and void. A user can choose an arbitrary name for a user-defined function and the name should follow the same rule for identifiers. An argument list can contain a number of arguments separated by a comma. An argument should be declared as a type followed by its name.

```
int foo(pdf p, int x);
```

The example above is the declaration of function foo, whose return type is an int and which takes a pdf type variable as the first argument and an int variable as the second argument.

Function Definitions

Definitions for user-defined functions should follow the start function. Thus, the order between functions is:

```
<user-defined function declarations>
<start function>
<user-defined function definitions>
```

Function definitions follow the same format as function declarations in addition to a body block, which are a set of statements executed when the function is called. However, the argument names can be different from the names used in function declarations as long as the type, number, and order of arguments match with function declarations.

```
<return type> <function name> (argument list) {body}
```

If the function definition for foo function used as an example for function declaration above is the following, the foo function will increment the value of the second argument and return the value.

```
int foo(pdf p, int x);
{
   x = x+1;
   return x;
}
```

Function Calls

User-defined functions can be called in any function including the calling function itself , allowing recursive calls. A sound function invocation matches the name, number and type of arguments exactly as for the function declaration. No name overloading is allowed. Also, to call a function, the function should be declared in advance. When a function is called with a list of arguments, the arguments are evaluated before calling the function from left to right. When the function returns a value, the returned value can be retrieved. However, this is optional. Thus, the function call follows the notation below.

```
(<variable> =)? <function name> (argument list);
```

Below are presented two examples of correct invocation of the function foo declared in 4.2:

```
pdf p;
p = “file.pdf”;
```

```
int y;
y = 1;
foo(p, y);
int r;
r = 3;
r = foo(p, y);
```

Argument Evaluation

Function arguments are evaluated following the *applicative approach:* left to right before the execution of the function.

3.7 Scope

A program should be compiled at once which means that all source code should be in one file. Thus, there is one kind of scope in SPML, the lexical scope of an identifier, that is the region of a program in which the identifier is recognized.

Lexical Scope

In SPML, there is one name space available for variable and function names. To clarify the explanation of the lexical scope, the meaning of a block should be defined. A block is a set of statements grouped by curly brackets, {}. Thus, a block of statements is executed like a single statement syntactically. A block can be declared only as a body of function, while, and if…else statements.

An identifier can be declared once in a block within a name space. Identifiers can be used several times in its block. If the same name of an identifier is declared outside the block, the identifier in the outer block will be undermined in the current block. Thus, SPML is following static scoping, which means the life of an identifier begins with its declaration and ends at the end of its block.

4. Project Plan

4.1 Processes Used

Since our group was formed, we have had regular meetings every Monday to set the next milestone for our project and to share the progress made after each week. Having weekly meetings from the early stage was very efficient and helpful for our project since all members became aware of the status of the project and planned for the following week as a group. For planning, we first needed to decide which language we would implement. After several discussions among group members and suggestions from the professor and the TA, our group chose a PDF manipulation language and started specification of the project while working on project proposal and language reference manual. Especially we could complete the specification of our language while writing the language reference manual. Then, we began developing our language according to the language reference manual according to the assignments of the language implementation to each group member. Our group made sure that different parts of the implementation were developed (at least familiarized) simultaneously so that we could finish the project on time. Also, we started documentation with implementation for better outcome at the end. By the time we focused developing code generation, we started testing.

4.2 Programming Style Guide

This is the style guide used while implementing SPML. This guide is based on Java Code Conventions available at http://java.sun.com/docs/codeconv/.

Naming

All classes and ANTLR related files will be declared in spml package.

Indentation and Spacing

Each block of code should be indented using a tab or white spaces while lines in the same block are indented in the same way.

Comments

Each class and method will be explained according to Javadoc format in the

beginning. Writing comments in the beginning of each meaningful block of code are strongly recommended.

4.3 Project Timeline

The following is the timeline for project milestones we had for SPML.

2-7-2007	Project Proposal completed
3-5-2007	Language Reference Manual completed
3-21-2007	Lexer/Parser completed
5-1-2007	Semantic Checking of the tree walker completed
5-1-2007	Testing Started (Test suite completed)
5-6-2007	Code generation of the tree walker completed
5-6-2007	Coordination with PDF API (iText library) completed
5-6-2007	Testing Completed

4.4 Roles and Responsibilities

Our team consists of one group leader and three other members. The leader was responsible for scheduling and project management in addition to development. For the project proposal and the language reference manual, all members participated in completing them. After we finished the language reference manual, we assigned each team member primary responsibilities for efficient development and timely completion. We followed this assignment until the completion of our project.

- Stefano Pacifico
  - Group Leader
  - Code Generation
  - Testing

- Jayesh Kataria
  - Semantic Checking Tree Walker
  - Code Generation Tree Walker
  - Code Generation

- Dhivya Khrishnan
  - Lexer
  - Parser
  - Semantic Checking Tree Walker

- Hye Seon Yi
  - Setting up CVS
  - Lexer
  - Parser
  - Documentation

4.5 Software Development Environment

Implementation Language: Java 5.0
Lexer, Parser, and Tree walker: ANTLR 2.7.6
Version Control: CVS
IDE (Integrated development environment): Eclipse 3.2.2 with ANTLR plug-in

4.6 Project log

1-23-2007	Project team formed
1-24-2007	Group meeting for language brainstorming
1-29-2007	Group meeting for language brainstorming II
2-1-2007	CVS installed for version control
2-2-2007	Group meeting for language brainstorming III
2-5-2007	Language chosen for the project (a PDF manipulation language)
2-5-2007	Group meeting for language features and project proposal
2-7-2007	Project Proposal completed
2-12-2007	Group meeting for deciding language grammar
2-15-2007	Assignment of language implementation responsibilities to each group member
2-15-2007	Abhi assigned for a mentor of our group
2-19-2007	Group meeting & division of work for Language Reference Manual

	(LRM)
2-26-2007	Group meeting for LRM
2-27-2007	Lexer/Parser first draft done
3-1-2007	LRM first draft done
3-5-2007	Group meeting for LRM II
3-5-2007	LRM completed
3-12-2007 ~ 3-16-2007	Spring break
3-14-2007	Group meeting for Lexer/Parser and grammar
3-17-2007	Final report cover done
3-17-2007	Group meeting for implementing arithmetic expression up to code generation
3-21-2007	Lexer/Parser completed
3-25-2007	Group meeting for language grammar
3-25-2007	Building AST in parser completed
3-29-2007	Group meeting for language grammar & tree walker
4-2-2007	Group meeting for language grammar
4-9-2007	Group meeting for tree walker and implementation
4-16-2007	Group meeting
5-1-2007	Semantic Checking of the tree walker completed
5-1-2007	Testing Started (Test suite completed)
5-6-2007	Code generation of the tree walker completed
5-6-2007	Coordination with PDF API (iText library) completed
5-6-2007	Testing Completed

5. Architectural Design

SPML is a compiler language which consists of four phases namely Front End, Tree Walker, Back End, and Runtime Libraries. Each phase is composed of one or more classes which are listed in the figure below.

Front End	
Class	Filename
SPMLLexer	lexerparser.g

SPMLParser	
SPMLTokenTypes	

Tree Walker		
Group	Class	Filename
Semantic Checking	SPMLTreeWalker	SpmlWalker.g
	SPMLTreeWalkerTokenTypes	
	CompilerException	CompilerException.java
Code Generation	SPMLCodeGen	SpmlCodeGen.g
	SPMLCodeGenTokenTypes	
	SpmlOut	SpmlOut.java
Environment	Env	Env.java
	SymbolTable	SymbolTable.java
	SpmlAST	SpmlAST.java
Type	Type	Type.java
	TypeArray	TypeArray.java
	TypeInt	TypeInt.java
	TypePDF	TypePDF.java
	TypeString	TypeString.java
	TypeVoid	TypeVoid.java
	Prototype	Prototype.java
Variable	Variable	Variable.java
	Iden	Iden.java
Others	PDFItem	PDFItem.java
	Main	Main.java

Back End	
CodeGen	CodeGen.java
SPMLLibrary	SPMLLibrary.java

Runtime Libraries	
JRE System Library	Multiple
iText Library	itext.jar
XPAAJ Library	XPAAJ.jar

Front End

In Front End, SPML source code is taken as an input and an AST tree is generated at the end as an output. There are two classes related to Front End namely SPMLLexer and SPMLParser and they are generated by ANTRL from lexerparser.g.

- SPMLLexer

SPMLLexer constructs a number of tokens from a stream of characters of SPML source code. Comments and whitespace are ignored whereas identifiers and symbols are recognized as tokens. When an undefined character or combination of characters is in source code, one or more lexical errors will be reported.

- SPMLParser

SPMLParser constructs an AST tree with the tokens generated from SPMLLexer using the defined grammar rules. When an undefined combination of tokens is in source code, one or more parsing errors will be reported.

Tree Walker

In Tree Walker, static semantic analysis and code generation are performed on the AST tree generated from Front End. There are many classes and files related to SPML tree walker, so grouping is used to explain them efficiently

- Semantic Checking: The classes in this group are related to static semantic checking of SPML language.

  - SPMLTreeWalker

This is where static semantic checking is performed. SPML supports extensive static semantic checking and below is the checkpoints of the analysis. When any of these checkpoints are violated, one or more semantic errors will be reported.

- ✓ A variable is used without a declaration.
- ✓ A variable is used without an initialization. Even when an initialization of a variable is done inside of a different block, the variable is still recognized as uninitialized. In the example below, `a` is recognized as uninitialized since it is initialized in a different block (if statement in this case).

```
start(){
   int a;
   if (condition){
      a = 1;
   }
   else{
      a = 0;
   }
   return a; /* error: a is uninitialized */
}
```

- ✓ A non-array variable is used like an array with an index.
- ✓ There is a type mismatch in expressions, such as arithmetic, conditional, and logical expressions. For example, when a string is used in logical expressions as an operand, a type mismatch error will be thrown.
- ✓ There is a type mismatch in operators, such as unary, multiplicative, additive, relational, equality, and logical operators. For example, when a pdf variable is used as an operand in a relational operator, a type mismatch error will be thrown.
- ✓ There is a type mismatch in statements, such as assignment, PDF, conditional, iterative, print, and jump statements. For example, when an integer value is assigned to a pdf variable, a type mismatch error will be thrown.
- ✓ Jump statements such as continue and break are used in a non-while block.
- ✓ A return statement is used in start().
- ✓ There are unreachable code after a return statement.

- ✓ A function is used without a function prototype. For A user function to be used by other functions including start(), it should be declared with a function prototype before start() and it should be defined with a body after start().
- ✓ A function definition exists without a function prototype. This case generates a semantic error if the function is called by other functions or not.
- ✓ Function arguments are not matched in a function prototype and definition. The number, types, order of arguments should be matched between a function prototype and its function definition.
- ✓ A user function does not have return statements except when the function is void.
- ✓ A function or identifier is redefined. One identifier should be used as one meaning in one block.

➢ SPMLTreeWalkerTokenTypes

This is created by ANTLR from SpmlWalker.g.

➢ SpmlAST

This counts line numbers of SPML code for reporting line numbers of semantic errors.

➢ CompilerException

This function throws semantic errors according to a string passed.

- Code Generation: The classes in this group are related to generating Java output code from SPML source code.

➢ SPMLCodeGen

This class is generated by ANTLR from SpmlCodeGen.g. This class calls CodeGen class to generate output code.

- SPMLCodeGenTokenTypes

  This is created by ANTLR from SpmlCodeGen.g.

- SpmlOut

  This class holds Java output code generated from SPML compiler.

- Environment: The classes in this group are used to maintain information about symbols in the current scope.

  - Env

    This contains a stack of symbol tables of variables and function prototypes of the source code in SPML. A symbol table is created for each block. This class contains methods to manipulate a symbol table as well.

  - SymbolTable

    This represents a symbol table in SPML.

- Type: The classes in this group

  - Type

    This is an abstract class for types in SPML

  - TypeArray

    This represents an array of any type (int, string, and pdf) in SPML.

  - TypeInt

    This represents int type in SPML.

  - TypePDF

This represents pdf type in SPML.

- TypeString

This represents string type in SPML.

- TypeVoid

This represents void type in SPML.

- Prototype

The class represents a function prototype.

- Variable
  - Variable

This represents a variable in SPML. A variable has a name and a type.

  - Iden

The class represents an identifier in SPML.

- Others
  - PDFItem

The class represents PDF objects in Java output code.

  - Main

The class represents PDF objects in Java output code.

Back End

- CodeGen

This has methods generating appropriate code for each statement while walking the AST.

- SPMLLibrary

This library contains all PDF related functionalities which utilize iText and XPAAJ API libraries internally. Thus, SPMLLibrary should be used to compile and run the translated Java output code from Tree walker. By having SPMLLibrary, SPML provides a layer of abstraction in the language design. When external libraries are changed or modified, only SPMLLibrary needs to be changed not the grammar.

Runtime Libraries

- JRE System Library

JRE System Library consists of core executables and files for standard Java platform. This library is required to run any SPML code since SPML code will be translated into Java output code.

- iText Library (Official site: http://www.lowagie.com/iText/)

This is an external library to support PDF functionalities. The majority of PDF functionalities for SPML are depending on iText API.

iText is an Open Source library developed by Bruno Lowagie and Paulo Soares for generating and manipulating PDF files. There are three main advantages in using iText library for developing SPML language. First, iText library contains extensive functionalities related to PDF files even though it is free to be used by everyone. Second, it provides APIs written in Java which is the target language for SPML. This made SPML implementation for PDF features seamless and convenient. Lastly, the time to implement PDF features in SPML language was reduced significantly. Without the use of external libraries, it would have been

impossible to develop SPML language in one semester. Moreover, this allowed our team to focus more on language features of SPML than implementation purely in Java.

- XPAAJ Library (Official site: http://www.adobe.com/devnet/livecycle/downloads/xpaaj.html)

XPAAJ (XML/PDF Access API for Java) is a Java API developed by Adobe which allows to create PDF files and extract context from them[5].

6. Test Plan

To testing SPML language, one shell script is created to test SPML Lexer / Parser / Tree Walker automatically. The script reads a set of SPML source code to verify a part of SPML language. The script made testing much easier since only thing needed is adding a new SPML file to be tested.

In addition to automatic testing, several SPML source programs are written for manual error checking. Especially our team found that manual checking is more efficient and accurate for testing for code generation and output results.

- Front End Testing (Lexer/Parser checking) & Tree Walker Testing (Semantic checking)

This tests two things at once: First, whether SPMLLexer and SPMLParser takes SPML input code and outputs an AST according to the SPML grammar defined by the language reference manual. Second, whether SPMLTreeWalker takes an AST and performs semantic checking correctly. The script is named as test-walker.sh.

- Code Generation Testing (Output code checking) & Back End Tesing

This tests are verifying the Java output code and runtime results, which is tested manually for efficiency and accuracy. A set of SPML programs are written to test code generation and back end parts of SPML.

---

[5] Mike Potter, " Developing applications with XPAAJ". http://blogs.adobe.com/mikepotter/XPAAJ_tutorial.pdf

7. Lessons Learned

- Stefano Pacifico

Creating a programming language and its compiler, is much more than one can think of. It not only requires knowledge of compilers concepts, but also great amounts of software design savvy, programming skill, communication prowess and good organization. That is the first thing that I have realized.

It is to be remarked how important, and difficult, the initial design turned out to be. We have made decisions at the beginning, which appeared to be reasonable, in order to not increase the complexity of the language. Some of them, such as not to include a boolean type, resulted in a major increase of complexity in the development of the compiler. Careful understanding of the design process, and thorough knowledge of the development environment with its implication are crucial.

Working intensely on the code generation and testing parts, increased my coding skills, and as the leader of the group, showed me how critical are the interactions among heterogeneous modules in a project.

Also as the project manager , I had to oversee the entire development process, thus getting familiar with almost all the aspects of the project. I have also realized the importance of communication and of team organization. The use of email, CVS, and VoIP has helped, though I believe, that also other collaborative tools could have been exploited. In particular the use of a project wiki, or a blog, could have been great resources.

Lastly, it is very rewarding to work with other people on a common goal and seeing it getting accomplished after many efforts.

- Jayesh Kataria
  - Technical Lessons

- Since I have always coded in Java, it was a new experience to code the Static Semantic Checker and the Code Generator in Antlr. It was especially interesting to combine Antlr with Java and figure out what was the best method to do so.

- I am not a big proponent of recursion but working with Antlr code made me realize the beauty of recursion and opened my mind to thinking and solving problems recursively.

- I generally develop code using the Java Bean methodology which has getter and setter methods. But I realized the importance of using immutable structures in this project after finding out that my code was inadvertently changing some properties of variables because they were mutable.

- Last but not the least, it is not every time that one gets to build compiler and working in the project has greatly improved my concepts in Complier Design. Looking back I can see which design decisions were not optimal and this will surely help me in my future projects.

■ Teamwork Lesson

- This is the first time I got a chance to work in a team with people from different cultures from my own and very different mindsets. This taught me to listen to every idea that was put forward with an open mind and learn from everyone.

- Of course, it is rare to find a team with no issues and ours was no exception. But patience and mutual understanding from everybody ensures that ultimately everything was resolved.

- I realized the importance of regular group discussions and meetings and I can surely say that they went a long way in enabling us to accomplish our goal and finish our project.

- Dhivya Khrishnan

  - Technical

    - Coding in ANTLR for lexerparser.g and SPMLWalker.g was a lot of fun. I got to understand when to avoid non determinism errors and how to remove them if at all they appear.
    - Working in Lexer and Parser helped me understand the initial phases of compiler (tokens, syntactic errors etc…)
    - Also, in the process of writing the walker that checks for semantic errors, I got to completely understand the insides of a compiler. I now have a good idea of how the compiler catches errors (semantic ones). Applying the theoretical concepts learnt in class into the project was very helpful.

  In all, doing the project was a lot of fun and a great opportunity to thoroughly understand the compiler concepts.

  - Team Work Lessons

  We planned very well right from the beginning of the course. This was very helpful throughout. We had periodic meetings for the project and that was one of the major reasons for being on schedule. However, initially we had some problems coordinating between the members for common timings but things smoothened out very well and we were able to do productive work as a team.

- Hye Seon Yi

The first thing I learned is that implementing a language can be much easier when you use a right tool like ANTLR. This was very different from the painful previous experience in my undergraduate, which was developing a parser in C from the scratch, which took so much time and pain from me. It was very exciting to see a lexer and a parser working, which was my primary responsibility, in a week even though I fixed them later on.

The second lesson is the importance of communication. Since everyone has a different work and even communication style, communication could boost or

threaten the completion and success of a project especially when members are working on the same project for the first time. Also, sharing each other's progress and updates always help project members get involved in the project more actively.

The third lesson is starting the project "early", scheduling "well" from the start, and sticking to the schedule "hard". I think this is a key for any project with several members especially when the members have different responsibilities and time tables. Also, this will allow to have extra time to fix any problems during development especially when something unexpected thing happened, such as a critical error in the design and so on.

In conclusion, this experience allowed me to use a useful tool like ANTLR and to learn key success factors in application development. Moreover, I am so glad and proud that our project was a very successful one at the end.

8. Appendix

Front End

lexerparser.g

SPML grammar for Lexer / Parser

```
// Lexer
//@ author :  Hye Seon , Dhivya
header{
   package spml;
}

class SPMLLexer extends Lexer;
options {
   k = 2;
   testLiterals = false;
   exportVocab = SPML;
   charVocabulary = '\3'..'\377';
```

```
}

// punctuation
LPAREN : '(';
RPAREN : ')';
LCURLY : '{';
RCURLY : '}';
LBRACKET : '[';
RBRACKET : ']';
COMMA : ',';
DQUOTE : '"';
SEMI! : ';';

// string constant
STRCON : '"' (~('"' | '\n'))* '"' ;

// operators
NOT : '!';
PLUS : '+';
MINUS : '-';
UMINUS : '~';
TIMES : '*';
DIV : '/';
ASSIGN : '=';
EQ : "==";
INEQ : "!=";
AND : "&&";
OR : "||";
GT : '>';
GEQ : ">=";
LT : '<';
LEQ : "<=";
COLON : ':';

// identifiers
ID options {testLiterals = true;}
```

```
: LETTER (LETTER | DIGIT | '_')*;

// number
NUMBER : (DIGIT)+;

protected LETTER : ('a'..'z' | 'A'..'Z');
protected DIGIT : '0'..'9';

NEWLINE : ( '\n' | '\r'
          | ('\r' '\n') => '\r' '\n')
          {newline(); $setType(Token.SKIP);};

WS : ( ' ' | '\t' )
     { $setType(Token.SKIP); };

// comment
MLCOMMENT : "/*" (options {greedy = false;} : .)* "*/"
            { $setType(Token.SKIP); };
// Parser
class SPMLParser extends Parser;

options {
   k = 2;
   buildAST = true;
   exportVocab = SPML;
}

tokens {
  CODE;
  FUNC_PROTOS;
  FUNC_BODY;
  DECLS;
  ARGS;
  ARG;
  OPSTMTS;
  CONTROLSTMT;
```

```
  ELSESTMT;
}

// main function statement
program :(func_prototypes)* main (function)* EOF!
          { #program = #([CODE], program); }// main root of our tree
        ;

main : "start"^ LPAREN! RPAREN! body
     ;

body : LCURLY! (stmt)* RCURLY   //!  all scopes will end with a } in
   the tree
     ;

// expressions
expr :   conditional_expr ( AND^ conditional_expr
                         | OR^ conditional_expr)*
                         ;

conditional_expr : plus_minus_expr ( GT^ plus_minus_expr
                                   | GEQ^ plus_minus_expr
                                   | LT^ plus_minus_expr
                                   | LEQ^ plus_minus_expr
                                   | EQ^ plus_minus_expr
                                   | INEQ^ plus_minus_expr)*
                 ;

plus_minus_expr : times_div_expr ( PLUS^ times_div_expr
                                 | MINUS^ times_div_expr)*;

times_div_expr : primary_expr ( TIMES^ primary_expr | DIV^
   primary_expr )*
                   ;

primary_expr : (NOT^|UMINUS^)? ((ID (array_expr)?) | NUMBER)
```

```
              | LPAREN! expr RPAREN!
              | function_call
              ;

array_expr : LBRACKET plus_minus_expr RBRACKET
           ;

blank_array : LBRACKET RBRACKET
            ;

// function statement
func_prototypes : ("void"|type(blank_array)?) ID functionprog SEMI!
{ #func_prototypes = #([FUNC_PROTOS], func_prototypes); }
                ;
function : ("void"|type(blank_array)?) ID functionprog body
{ #function = #([FUNC_BODY], function); }
         ;

functionprog : LPAREN! func_paras RPAREN!
{ #functionprog = #([ARGS], functionprog); }
             ;

func_paras : (param (COMMA! param)*)?
              ;

param: type ID (blank_array)?
{ #param = #([ARG], param); }
              ;
type : "int" | "string" | "pdf"
     ;

function_call : ID LPAREN (((expr|STRCON) (COMMA! (expr|STRCON))*)?)
   RPAREN
   ;

function_call_stmt : function_call SEMI!
```

```
                ;

// type statements
int_stmt : "int" ID int_tail SEMI!
         ;
int_tail : (decl_array)?
                (COMMA! ID (decl_array)?)*
         ;

decl_array : LBRACKET NUMBER RBRACKET
;

// only pdf has blank array assignments
pdf_stmt : "pdf" ID pdf_tail SEMI!
         ;
pdf_tail : (decl_array)?
             (COMMA! ID (decl_array)?)*
         ;

string_stmt : "string" ID string_tail SEMI!
            ;
string_tail : (decl_array)?
                (COMMA! ID (decl_array)?)*
            ;

decls_stmt:decl_stmt
{#decls_stmt = #([DECLS], decls_stmt);}
  ;

decl_stmt : int_stmt
          | pdf_stmt
          | string_stmt
          ;

// operation statements
create_stmt : "create"^ (STRCON|ID (array_expr)?|function_call)
```

```
            ;

highlight_stmt : "highlight"^ (ID(array_expr)?|function_call )
   (STRCON|ID(array_expr)?|function_call)
                ;
extractpage_stmt : "extractpage"^ (ID(array_expr)?|function_call)
   (numid | function_call)
                  ;
set_stmt : "setauthor"^ (ID(array_expr)?|function_call)
   (STRCON|(ID(array_expr)?)|function_call)
          ;
get_stmt : "getauthor"^ (function_call | ID(array_expr)?)
          ;

op_stmts : (highlight_stmt SEMI! // removed close_stmt
          | set_stmt SEMI!
          | print_stmt SEMI!
          | totext_stmt SEMI!)
{#op_stmts = #([OPSTMTS], op_stmts); }
           ;

// other statements
conditional_stmt : "if"^ LPAREN! expr RPAREN! (body) (elsepart)?
{#conditional_stmt = #([CONTROLSTMT], conditional_stmt); }
                   ;

elsepart: (options {greedy=true;} : "else"^ (body))
{ #elsepart = #([ELSESTMT], elsepart); }
                  ;
iteration_stmt : "while"^ LPAREN! expr RPAREN! (body)
{#iteration_stmt = #([CONTROLSTMT], iteration_stmt); }
                ;
jump_stmt : "return"^ (expr|STRCON)? SEMI!
           | "continue"^ SEMI!
           | "break"^ SEMI!
           ;
```

```
length_stmt: "length"^ (ID(array_expr)?|function_call)
          ;
print_stmt : "print"^
   ((expr)=>(expr)|(strcon_expr)=>(strcon_expr)|STRCON|(ID(array_expr)
   ?)|function_call)
          ;

totext_stmt : "totextfile"^ (ID(array_expr)?|function_call)
   (STRCON|(ID(array_expr)?)|function_call)
            ;

in_stmt : (STRCON|(ID(array_expr)?)|"pdf"|function_call) "in"^
   (STRCON|ID(array_expr)?|function_call)
         ;

strcon_expr : (
                  STRCON
                | NUMBER
                | (ID (array_expr)?)
                | function_call
               )
               (((PLUS^ (
                         STRCON
                        | NUMBER
                        | (ID (array_expr)?)
                        | function_call
                       )
                 )*)
               |(EQ^|INEQ^)(STRCON
                        | NUMBER
                        | (ID (array_expr)?)
                        | function_call
                       ))

               ;
```

```
id_assign_stmt : (ID|(ID LBRACKET plus_minus_expr RBRACKET))
                  ASSIGN^
                 (   (in_stmt) => in_stmt
                   |(expr) => expr
                   | (strcon_expr) => strcon_expr
                        /* for string manipulation */
                   | get_stmt
                   | extractpage_stmt
                   | length_stmt
                   | create_stmt
                   //| in_stmt
               ) SEMI!
               ;
numid : ID(array_expr)?
     | NUMBER
      ;

// all statements
//id_assign_stmt has some of the pdf statements
//   - cause they need the assign operator in them
stmt : id_assign_stmt
     | decls_stmt
     | conditional_stmt
     | iteration_stmt
     | jump_stmt
     | op_stmts
     | function_call_stmt
     ;
```

Tree Walker

CompilerException.java

Generate semantic errors from Tree Walker

```
package spml;

/** This class is used to report errors during the compilation
 * @author stefy
 *
 */
public class CompilerException extends Exception {
   CompilerException(String message){
        super(message);
   }
}
```

Env.java

Maintain environmental objects such as symbol tables and provide methods to manipulate symbol tables

```
/**
 * @author Jayesh and Dhivya
 */
package spml;

import java.util.Enumeration;
import java.util.Stack;
import java.util.Hashtable;

public class Env {

   Hashtable<String,Prototype> funcProtos;

   Stack<SymbolTable> stackTable; /* This is a stack of all symbol
   tables. TOS is the most current scope*/

   public Env(){
        funcProtos = new Hashtable<String,Prototype>();
        stackTable = new Stack<SymbolTable>();
   }
```

```
  public void addProto(Prototype p){
      if(p == null || p.getProtoName() == null || p.getProtoName() ==
  ""){
            System.out.println("Invalid parameters in Prototype");
            return;
      }
      //TODO: throw exception here
      funcProtos.put(p.getProtoName(), p);
  }

  public Prototype getProto(String name){
      return funcProtos.get(name);
  }

  public void refreshProto(Prototype p){
      addProto(p);
  }

  public boolean protoExists(Prototype p){ //checks if the return
  type, name and args list of the function matches the prototype
//     boolean result = false;
//     System.out.println("Checking func "+p.getProtoName());
       Prototype in = getProto(p.getProtoName());
       if(in == null)
             return false;
//     System.out.println("Proto name exists. Now checking return
  type:");

       if(!p.getReturnType().getType().equals(in.getReturnType().getTy
  pe()))
             return false;
//     System.out.println("Return type matched. Now checking number of
  args");
       if(p.getArgSize()!=in.getArgSize())
             return false;
```

```
//     System.out.println("Length of args matched. Now checking each
  arg type");
       for(int i=0; i<p.getArgSize(); i++){
              Variable ofp = p.getArgAtPosition(i);
              Variable ofin = in.getArgAtPosition(i);
//            System.out.println(ofp.getVarType().getType() + " vs "+
  ofin.getVarType().getType());

       if(!ofp.getVarType().getType().equals(ofin.getVarType().getType
  ()))
                     return false;
       }

       return true;

  }

  public void printAllProtos(){
       Enumeration<Prototype> e = funcProtos.elements();
       while(e.hasMoreElements()){
              Prototype p = e.nextElement();
              System.out.println(p.toString());
       }
  }

  public void addNewScope(){
       SymbolTable st;
       SymbolTable outer;
       if(stackTable.empty())
              outer = null;
       else
              outer = stackTable.peek();
       st = new SymbolTable(outer);
       stackTable.push(st);
//     return st;
  }
```

```
/*
   private void addNewScope(SymbolTable tab){

   }
   */

   public SymbolTable deleteCurrentScope(){
       if(!stackTable.empty())
              return stackTable.pop();
       return null;
   }

   public SymbolTable getCurrentScope(){
       if(!stackTable.empty())
              return stackTable.peek(); //debug
       return null;
   }

   /**
    * This is used so that when changes are made to the symbol table
    * retrieved using getCurrentScope, they also reflect in the env
   stack
    * @return
    */
   public void refreshCurrentScope(SymbolTable tab){ // tab is the
   refreshed table passed
       this.deleteCurrentScope();
       this.stackTable.push(tab);

   }
   /**
    *
    *
    */
   public void printAllScopes(){
       if(stackTable.empty()){
```

```
            System.out.println("No scopes present!!");
            return;
        }
        System.out.println("current size of stack =
    "+stackTable.size());
        for(int i = 0; i< stackTable.size();i++){
            System.out.println("Scope: "+i);
            SymbolTable t = stackTable.get(i);
            t.printTable();
            System.out.println();
        }
    }
}
```

Iden.java

Represent an identifier

```
/**
 * @author Jayesh and Stefano
 */

package spml;

public class Iden {
    private String name;

    public Iden(String n){
        this.name = n;
    }

    public String getName(){
        return name;
    }

    public void setName(String n){
        this.name = n;
```

```
    }

    public static Iden newInstance(Iden i){
        Iden result = new Iden(i.getName());
        return result;
    }
}
```

PDFItem.java

Represent PDF objects in Java output code

```
/**
 * @author Stefy
 */

package spml;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.FileOutputStream;
import java.io.IOException;

public class PDFItem {
    private File pdf_file;

    public PDFItem(){
        this.pdf_file = new
    File(CodeGen.TMPPREFIX+Long.toString(System.currentTimeMillis())+".
    pdf");
    }
    public PDFItem(String p){
        this.pdf_file = new File(p);
    }

    public PDFItem sum(PDFItem expr){
        return SPMLLibrary.concatenate(this.getPath(), expr.getPath());
```

```
    }

    public PDFItem sum(PDFItem expr, String out){
        return SPMLLibrary.concatenate(this.getPath(), expr.getPath(),
    out);
    }

    /*public PDFItem(PDFItem copy) throws IOException {
        FileOutputStream o = new FileOutputStream(this.pdf_file);
        FileInputStream i = new FileInputStream(new
    File(copy.getPath()));
        byte[] buf = new byte[1024];
         int len;
         while ((len = i.read(buf)) > 0) {
             o.write(buf, 0, len);
         }
         i.close();
         o.close();
    }
    */
    public String getPath(){
        return this.pdf_file.getAbsolutePath();
    }

    public File getPdfFile(){
        return pdf_file;
    }

}
```

Prototype.java

Represent a function prototype

```
/**
 * @author Jayesh and Dhivya
 */
```

```
package spml;

import java.util.ArrayList;

public class Prototype {
   private Type return_type;
   private ArrayList<Variable> arg_list;
   private String protoName;
   private boolean isdefined;
   private boolean isUsed;

   public Prototype(Type t, String name, ArrayList<Variable> list){
       this.return_type = t;
       this.arg_list = list;
       this.protoName = name;
       this.isdefined  = false;
       this.isUsed = false;
   }

   public Type getReturnType(){
       return return_type;
   }

   public ArrayList<Variable> getArgList(){
       return arg_list;
   }

   public void setReturnType(Type t){
       return_type = t;
   }

   public void setArgList(ArrayList<Variable> t){
       arg_list = t;
   }
```

```
public void addArgument(Variable t){
    arg_list.add(t);
}

public int getArgSize(){
    return arg_list.size();
}

public Variable getArgAtPosition(int position){
    if(position > this.getArgSize()-1)
          return null;
    return arg_list.get(position);
}

public String getProtoName() {
    return protoName;
}

public void setProtoName(String protoName) {
    this.protoName = protoName;
}
public boolean getIsdefined() {
    return isdefined;
}

public void setIsdefined() {
    this.isdefined = true;
}

public String toString(){
    StringBuffer res = new StringBuffer();
    res.append(return_type.getType()+" ");
    res.append(protoName+"( ");
    Variable v;
    int i;
```

```
        for(i=0; i<arg_list.size()-1;i++){
                v = arg_list.get(i);
                res.append(v.getVarType().getType()+" ");
                res.append(v.getVarName().getName()+", ");
        }
        if(i<arg_list.size()){

                        v = arg_list.get(i);
                        res.append(v.getVarType().getType()+" ");
                        res.append(v.getVarName().getName()+" ");

        }
        res.append(");");
        return res.toString();
    }

    public boolean isUsed() {
        return isUsed;
    }

    public void setUsed(boolean isUsed) {
        this.isUsed = isUsed;
    }
}
```

SpmlAST.java

Count line numbers of SPML source code for reporting line numbers of semantic errors

```
package spml;

import antlr.CommonAST;
import antlr.Token;
/**
 * @author JayeshKataria
 *
```

```
 */
public class SpmlAST extends CommonAST{
   /**
    *
    */
   private static final long serialVersionUID = -1936758840206008598L;
   // added by eclipse
   private int line=0;
   private int column = 0;

   /**
    * This is the constructor used to initialize the CommonAST Object
    *
    */
   public void initialize(Token tok){
       super.initialize(tok);
       this.line = tok.getLine();
       this.column = tok.getColumn();
   }

   public int getLine(){
       return this.line;
   }

   public int getColumn(){
       return this.column;
   }
}
```

SpmlCodeGen.g

Code generation grammar

```
//@author: Jayesh
header{
   package spml;
}
```

```
{
   import java.util.*;
}

class SPMLCodeGen extends TreeParser;
options {
   importVocab = SPML;
   defaultErrorHandler = true;
}

{
SpmlOut out = new SpmlOut();
boolean declFlag = false;
boolean controlFlag = false;
boolean returnFlag = false;
boolean hasMoreCode = false;
Type curRetType = null;
ArrayList<Type> tempForFunc = new ArrayList<Type>(1);
Env env = new Env();
boolean pdfKw = false;
boolean condFlag = false; // true when an if conditional is being
   walked and false when a assign statement is being passed. To
   differentiate between
                                    // while(a && b)   and c = a&&b
}

program throws CompilerException
{
// out.append("package spml;\n");
   out.append("import java.io.*;\n");
   out.append("import spml.*;\n");
   out.append("import java.io.FileInputStream;\n");
   out.append("import java.io.FileOutputStream;\n\n");
   out.append("import java.util.ArrayList;\n");
   out.append("import java.util.ListIterator;\n");
   out.append("import com.adobe.pdf.PDFDocument;\n");
```

```
   out.append("import com.adobe.pdf.PDFFactory;\n");
   out.append("import com.adobe.pdf.PDFWord;\n");
   out.append("public class "+SpmlOut.fileName+"{\n");

}
: #(CODE
       (func_protos)
       (mainBody) {
              // start closes here. So destroy the current scope.
//            env.deleteCurrentScope();
        }
       (func_body)
       {
             out.append("}");
       }
   )
   {}
;


mainBody throws CompilerException
{
   Variable dummy = null;
   String jump;
}
: #("start" {
       out.append(CodeGen.mainStmt());
       env.addNewScope();// adds a new Symbol Table for the current
   scope
   }
   mainBody RCURLY
   {env.deleteCurrentScope();out.append(CodeGen.deleteTmpFileGen());ou
   t.append(";\n}\n\n");})
| #( DECLS {declFlag = false;}declHandler {out.append("");{declFlag =
   false; }} ) mainBody
| #( ASSIGN
```

```
    {
        String lhsId = null ;
        Type rhsType = null;
        Type lhsType = null;
        SymbolTable cur = env.getCurrentScope();
        String arrExpr=null;
        Variable rExpr = null;
    }
    lhsId = strdecl (LBRACKET arrExpr = checkArrayExpr)?
    rExpr = rhsOfAssign
    {
        Variable lhs = Variable.newInstance(cur.get(lhsId));
        if(arrExpr!=null){ // indexing has been done hence check for
    arrays
//                  if(lhsType.getType().startsWith(Type.INT))
//                        lhsType = new TypeInt();
//                  if(lhsType.getType().startsWith(Type.PDF))
//                        lhsType = new TypePDF();
//                  if(lhsType.getType().startsWith(Type.STRING))
//                        lhsType = new TypeString();

              lhsId = lhsId + "["+arrExpr+"]";


        }
        if(lhs.getVarType().getType().equals(Type.PDF) &&
    rExpr.getVarType().getType().equals(Type.STRING)){
              // pdf p1; p1 = "DSfdssdfsd" ; this opens an existing
    pdf
              out.append(CodeGen.pdfOpen(new StringBuffer(lhsId),new
    StringBuffer(rExpr.getVarName().getName())));
//                  out.append(CodeGen.assignment(new
    StringBuffer(lhsId),new
    StringBuffer(rExpr.getVarName().getName())));
        }
        else{
```

```
            if(lhs.getVarType().getType().equals(Type.PDF) &&
   rExpr.getVarType().getType().equals(Type.PDF)){
                    //pdf = pdf needs to be handled separately
                    out.append(CodeGen.pdfAssignment(new
   StringBuffer(lhsId),new
   StringBuffer(rExpr.getVarName().getName())));
            }else{
            out.append(CodeGen.assignment(new
   StringBuffer(lhsId),new
   StringBuffer(rExpr.getVarName().getName())));
            }
        }
        out.append(CodeGen.getEndStmt());
   }

   )   { }
   mainBody   // end ASSIGN

|#(OPSTMTS opstmts)mainBody
|#(CONTROLSTMT controlstmts ) mainBody
|(dummy = getId){out.append(new
   StringBuffer(dummy.getVarName().getName()));out.append(CodeGen.getE
   ndStmt());} mainBody // this is for independent system calls
|jump = jump_stmts {

if(!controlFlag)
{
   //throw new CompilerException("Statement " +jump+" is wrongly used
   here, allowed only in while block");
}

} mainBody
|return_stmt //mainBody
| //nothing
;
// this is to check that return does not come in start
```

```
return_stmt throws CompilerException
{
   Variable curRetVar = null;
}

:#("return" {out.append("return "); returnFlag = true ; }
{
}
 // ok, the function's ret type is set in the global variable.
 (curRetVar = rhsOfAssign)?
 {

   if(curRetVar!=null){
        curRetType = curRetVar.getVarType();
        out.append(curRetVar.getVarName().getName());
   }
    else
        curRetType = null;
   out.append(CodeGen.getEndStmt());
   if(curRetType==null)
   {
        curRetType = new TypeVoid();
   }
   else
  {System.out.println("\n Global is "+curRetType.getType());}

 })  checkformore
;

checkformore throws CompilerException
:
| { hasMoreCode = false;}
|mainBody { hasMoreCode = true; }
;

//------------control stmts (if and while) start here----------------
```

```
   ----------------------------------------------------------------

controlstmts throws CompilerException
{
// Type t=null;
   Variable t = null;
   String jump;
}
:#("if"
  {
        // adding new scope
        env.addNewScope();
        condFlag = true;
  }
 (t=rhsOfAssign)
{
   if(t.getVarType().getType().startsWith(Type.PDF) ||
   t.getVarType().getType().startsWith(Type.STRING)){
       out.append("\n/*Got:"+t.getVarType().getType()+"*/\n");
       out.append(CodeGen.ifStmt(new
   StringBuffer(t.getVarName().getName())));
   }else{
       out.append(CodeGen.ifStmt(CodeGen.intExprEval(new
   StringBuffer(t.getVarName().getName()))));
   }
   condFlag = false;
}
  mainBody
   RCURLY{out.append("}\n");env.deleteCurrentScope();}(elsebody)?)


|#("while"
  {
        // adding new scope
//        controlFlag = true;
        env.addNewScope();
```

```
        condFlag = true;

  }
  (t=rhsOfAssign)
  {
   out.append(CodeGen.whileStmt(CodeGen.intExprEval(new
   StringBuffer(t.getVarName().getName()))));
   condFlag = false;
  }
  mainBody RCURLY {out.append("}\n");env.deleteCurrentScope();}
  )
{
   if(!t.getVarType().getType().equals(Type.INT)){
       //throw new CompilerException("Type mismatch in while
   conditional");
   }
}

//|#("else" {env.addNewScope();} )
;

jump_stmts returns[String j]
{
   j = null;
}
:"continue" { out.append("continue;\n"); j = "continue"; }
|"break" { out.append("break;\n"); j = "break" ; }
;

elsebody throws CompilerException
:#(ELSESTMT elsebody)
|#("else" {  env.addNewScope();out.append(CodeGen.elseStmt());}
   mainBody RCURLY {out.append("}\n");env.deleteCurrentScope();})
//|//mainBody
;
```

```
//------------control stmts (if and while) end here------------------
   ------------------------------------------------------------------
opstmts throws CompilerException
{
 Variable var = null;
 Variable var2 = null;

 Type ltype = null;
 Type rtype = null;
}
:#("highlight" var = getId var2=getId)
{
   out.append(CodeGen.highlight(new
   StringBuffer(var.getVarName().getName()),new
   StringBuffer(var2.getVarName().getName())));
   out.append(CodeGen.getEndStmt());
}
|#("setauthor" var = getId var2=getId)
{
   ltype = var.getVarType();
    rtype = var2.getVarType();
    if(!ltype.getType().equals(Type.PDF)
   || !rtype.getType().equals(Type.STRING)){
       //throw new CompilerException("Incompatible types
   "+ltype.getType()+","+rtype.getType()+" for setauthor statement");
   }
}
| #("print" printFunc)

|#("totextfile" var = getId var2=getId)
{
   out.append(CodeGen.totextStmt(new
   StringBuffer(var.getVarName().getName()),new
   StringBuffer(var2.getVarName().getName())));
   out.append(CodeGen.getEndStmt());
```

```
}
;

rhsOfAssign returns [Variable rexpr] throws CompilerException
{
  Variable var = null;
  Variable var2 = null;
  // constant = null;
  Type rtype = null;
  Type ltype = null;
  rexpr = null;
  Variable lhs = null;
  Variable rhs = null;//this is hack, the variable names will contain
  the entire expression and its type
  String fname = null;
  String l, r, varName;
  ArrayList<Type> funcArgTypes = null;
  //RHS can be: ID, Constant, Expression, Function, String.
    //Expression can be a combination of the above 4 types and a
  binary operator
}
:#(PLUS lhs = rhsOfAssign rhs = rhsOfAssign)
 {
  l = lhs.getVarName().getName();
  r = rhs.getVarName().getName();
  if(lhs.getVarType().getType().equals(Type.PDF)){
      varName = CodeGen.pdfSum(new StringBuffer(l),new
  StringBuffer(r)).toString();
  }
   else{
      if(condFlag
  && !(lhs.getVarType().getType().startsWith(Type.PDF)||lhs.getVarTyp
  e().getType().startsWith(Type.STRING))){
            varName = CodeGen.expressionGenIntToBool(new
  StringBuffer("("+l),new StringBuffer(r+")"),new
  StringBuffer("+")).toString();
```

```
        }
        else{
               varName = CodeGen.expressionGen(new
   StringBuffer("("+l),new StringBuffer(r+")"),new
   StringBuffer("+")).toString();
        }
    }
      rexpr = new Variable(Type.newInstance(lhs.getVarType()),new
   Iden(varName)); // dont know what will happen if arrays come into
   the pic
//     rexpr.append(CodeGen.expressionGen(lhs,rhs,new
   StringBuffer("+")));
 }
|#(MINUS lhs = rhsOfAssign rhs = rhsOfAssign)
{
   l = lhs.getVarName().getName();
   r = rhs.getVarName().getName();
   if(condFlag){
        varName = CodeGen.expressionGenIntToBool(new
   StringBuffer("("+l),new StringBuffer(r+")"),new StringBuffer("-
   ")).toString();
   }else{
        varName = CodeGen.expressionGen(new StringBuffer("("+l),new
   StringBuffer(r+")"),new StringBuffer("-")).toString();
   }
   rexpr = new Variable(Type.newInstance(lhs.getVarType()),new
   Iden(varName));
}
|#(TIMES lhs = rhsOfAssign rhs = rhsOfAssign)
{
   l = lhs.getVarName().getName();
   r = rhs.getVarName().getName();
   if(condFlag){
        varName = CodeGen.expressionGenIntToBool(new
   StringBuffer("("+l),new StringBuffer(r+")"),new
   StringBuffer("*")).toString();
```

```
   }else{
       varName = CodeGen.expressionGen(new StringBuffer("("+l),new
   StringBuffer(r+")"),new StringBuffer("*")).toString();
   }

   rexpr = new Variable(Type.newInstance(lhs.getVarType()),new
   Iden(varName));

}
|#(DIV lhs = rhsOfAssign rhs = rhsOfAssign)
{
   l = lhs.getVarName().getName();
   r = rhs.getVarName().getName();
   if(condFlag){
       varName = CodeGen.expressionGenIntToBool(new
   StringBuffer("("+l),new StringBuffer(r+")"),new
   StringBuffer("/")).toString();
   }else{
       varName = CodeGen.expressionGen(new StringBuffer("("+l),new
   StringBuffer(r+")"),new StringBuffer("/")).toString();
   }

   rexpr = new Variable(Type.newInstance(lhs.getVarType()),new
   Iden(varName));

}
|#(AND lhs = rhsOfAssign rhs = rhsOfAssign)
{
   l = lhs.getVarName().getName();
   r = rhs.getVarName().getName();
   if(condFlag){
       varName = CodeGen.expressionGen(new StringBuffer("("+l),new
   StringBuffer(r+")"),new StringBuffer("&&")).toString();
   }else{
       varName = CodeGen.expressionGenBoolToInt(new
   StringBuffer("("+l),new StringBuffer(r+")"),new
```

```
   StringBuffer("&&")).toString();
   }
   rexpr = new Variable(Type.newInstance(lhs.getVarType()),new
   Iden(varName));

}

|#(OR lhs = rhsOfAssign rhs = rhsOfAssign)
{
   l = lhs.getVarName().getName();
   r = rhs.getVarName().getName();
   if(condFlag){
       varName = CodeGen.expressionGen(new StringBuffer("("+l),new
   StringBuffer(r+")"),new StringBuffer("||")).toString();
   }else{
       varName = CodeGen.expressionGenBoolToInt(new
   StringBuffer("("+l),new StringBuffer(r+")"),new
   StringBuffer("||")).toString();
   }
   rexpr = new Variable(Type.newInstance(lhs.getVarType()),new
   Iden(varName));

}

|#(LT lhs = rhsOfAssign rhs = rhsOfAssign)
{
   l = lhs.getVarName().getName();
   r = rhs.getVarName().getName();
   if(condFlag){
       varName = CodeGen.expressionGen(new StringBuffer("("+l),new
   StringBuffer(r+")"),new StringBuffer("<")).toString();
   }else{
       varName = CodeGen.expressionGenBoolToInt(new
   StringBuffer("("+l),new StringBuffer(r+")"),new
   StringBuffer("<")).toString();
   }
```

```
   rexpr = new Variable(Type.newInstance(lhs.getVarType()),new
   Iden(varName));

}

|#(LEQ lhs = rhsOfAssign rhs = rhsOfAssign)
{
   l = lhs.getVarName().getName();
   r = rhs.getVarName().getName();
   if(condFlag){
       varName = CodeGen.expressionGen(new StringBuffer("("+l),new
   StringBuffer(r+")"),new StringBuffer("<=")).toString();
   }else{
       varName = CodeGen.expressionGenBoolToInt(new
   StringBuffer("("+l),new StringBuffer(r+")"),new
   StringBuffer("<=")).toString();
   }
   rexpr = new Variable(Type.newInstance(lhs.getVarType()),new
   Iden(varName));

}

|#(GT lhs = rhsOfAssign rhs = rhsOfAssign)
{
   l = lhs.getVarName().getName();
   r = rhs.getVarName().getName();
   if(condFlag){
       varName = CodeGen.expressionGen(new StringBuffer("("+l),new
   StringBuffer(r+")"),new StringBuffer(">")).toString();
   }else{
       varName = CodeGen.expressionGenBoolToInt(new
   StringBuffer("("+l),new StringBuffer(r+")"),new
   StringBuffer(">")).toString();
   }
   rexpr = new Variable(Type.newInstance(lhs.getVarType()),new
   Iden(varName));
```

```
}

|#(GEQ lhs = rhsOfAssign rhs = rhsOfAssign)
{
  l = lhs.getVarName().getName();
  r = rhs.getVarName().getName();
  if(condFlag){
      varName = CodeGen.expressionGen(new StringBuffer("("+l),new
  StringBuffer(r+")"),new StringBuffer(">=")).toString();
  }else{
      varName = CodeGen.expressionGenBoolToInt(new
  StringBuffer("("+l),new StringBuffer(r+")"),new
  StringBuffer(">=")).toString();
  }
  rexpr = new Variable(Type.newInstance(lhs.getVarType()),new
  Iden(varName));

}

|#(EQ lhs = rhsOfAssign rhs = rhsOfAssign)
{
  l = lhs.getVarName().getName();
  r = rhs.getVarName().getName();
  if(lhs.getVarType() instanceof TypePDF){
      if(condFlag){
            varName = CodeGen.pdfEqualityBool(new
  StringBuffer(l),new StringBuffer(r)).toString();
      }else{
            varName = CodeGen.pdfEqualityInt(new StringBuffer(l),new
  StringBuffer(r)).toString();
      }
  }
  else if(lhs.getVarType() instanceof TypeString){
      if(condFlag){
            varName = CodeGen.stringEqualityBool(new
```

```
   StringBuffer("("+l),new StringBuffer(r+")")).toString();
        }else{
              varName = CodeGen.stringEqualityInt(new
   StringBuffer("("+l),new StringBuffer(r+")")).toString();
        }
   }
   else{ // typeInt remains
        if(condFlag){
              varName = CodeGen.expressionGen(new
   StringBuffer("("+l),new StringBuffer(r+")"),new
   StringBuffer("==")).toString();
        }
        else{
              varName = CodeGen.expressionGenBoolToInt(new
   StringBuffer("("+l),new StringBuffer(r+")"),new
   StringBuffer("==")).toString();
        }
   }
   rexpr = new Variable(Type.newInstance(lhs.getVarType()),new
   Iden(varName));
}

|#(INEQ lhs = rhsOfAssign rhs = rhsOfAssign)
{
   l = lhs.getVarName().getName();
   r = rhs.getVarName().getName();
   if(lhs.getVarType() instanceof TypePDF){
        if(condFlag){
              varName = CodeGen.pdfInequalityBool(new
   StringBuffer(l),new StringBuffer(r)).toString();
        }else{
              varName = CodeGen.pdfInequalityInt(new
   StringBuffer(l),new StringBuffer(r)).toString();
        }
   }
   else if(lhs.getVarType() instanceof TypeString){
```

```
        if(condFlag){
              varName = CodeGen.stringInequalityBool(new
    StringBuffer("("+l),new StringBuffer(r+")")).toString();
        }else{
              varName = CodeGen.stringInequalityInt(new
    StringBuffer("("+l),new StringBuffer(r+")")).toString();
        }
    }
    else{ // typeInt remains
        if(condFlag){
              varName = CodeGen.expressionGen(new
    StringBuffer("("+l),new StringBuffer(r+")"),new
    StringBuffer("!=")).toString();
        }else{
              varName = CodeGen.expressionGenBoolToInt(new
    StringBuffer("("+l),new StringBuffer(r+")"),new
    StringBuffer("!=")).toString();
        }
    }
    rexpr = new Variable(Type.newInstance(lhs.getVarType()),new
    Iden(varName));
}

|#(UMINUS rhs = rhsOfAssign)
{
    r = rhs.getVarName().getName();
    varName = CodeGen.unaryMinus(new
    StringBuffer("("+r+")")).toString();
    rexpr = new Variable(Type.newInstance(rhs.getVarType()),new
    Iden(varName));
}

|#(NOT rhs = rhsOfAssign)
{
    r = rhs.getVarName().getName();
    if(condFlag){
```

```
        varName = CodeGen.unaryNotBool(new
   StringBuffer("("+r+")")).toString();
   }else{
        varName = CodeGen.unaryNotInt(new
   StringBuffer("("+r+")")).toString();
   }
   rexpr = new Variable(Type.newInstance(rhs.getVarType()),new
   Iden(varName));
}

|#("create" var = getId)
 {
   String vName = CodeGen.pdfCreate(new
   StringBuffer(var.getVarName().getName())).toString();
   rexpr = new Variable(new TypeInt(), new Iden(vName)); // it is not
   TypePDF here so that it works properly in assignment and does not
   call pdfAssignment
//     rexpr.setVarName(new Iden());
 }

|#("length" var = getId)
 {
   System.out.println("Type in length: "+var.getVarType().getType());
   String vName;
   if(var.getVarType() instanceof TypeArray){ //if it is an array of
   any type
        vName = CodeGen.arrayLength(new
   StringBuffer(var.getVarName().getName())).toString();
   }else{ // it can only be length pf pdf file i.e. num. of pages
        vName = CodeGen.pdfLength(new
   StringBuffer(var.getVarName().getName())).toString();
   }
   rexpr = new Variable(new TypeInt(), new Iden(vName)); // it is not
   TypePDF here so that it works properly in assignment and does not
   call pdfAssignment
//     rexpr.setVarName(new Iden());
```

```
 }

|#("extractpage" var = getId var2 = getId)
 {
   String str = CodeGen.extractStmt(new
   StringBuffer(var.getVarName().getName()),new
   StringBuffer(var2.getVarName().getName())).toString();
   rexpr = new Variable(new TypePDF(),new Iden(str));
  }

|#("in" ((var = getId)|"pdf"{pdfKw = true;}) var2 = getId)
{
   String vName=null;
   if(pdfKw){ //pdfs in "directory"
       pdfKw = false;
       vName = CodeGen.pdfInDirStmt(new StringBuffer("redundant"),new
   StringBuffer(var2.getVarName().getName())).toString();
   }else{
       if(var.getVarType() instanceof TypeString && var2.getVarType()
   instanceof TypeString){
             vName = CodeGen.randomNameGen("tmp");
             out.append(CodeGen.strInStrStmt(new
   StringBuffer(var.getVarName().getName()),new
   StringBuffer(var2.getVarName().getName()),vName));
             out.append(CodeGen.getEndStmt());
       }

       if(var.getVarType() instanceof TypeString && var2.getVarType()
   instanceof TypePDF){
             //string in pdf
             vName = CodeGen.stringInPdfStmt(new
   StringBuffer(var.getVarName().getName()),new
   StringBuffer(var2.getVarName().getName())).toString();
       }

       if(var.getVarType() instanceof TypePDF && var2.getVarType()
```

```
   instanceof TypeString){
              //pdf exists in Dir
              vName = CodeGen.pdfExistInDir(new
   StringBuffer(var.getVarName().getName()),new
   StringBuffer(var2.getVarName().getName())).toString();
        }
   }
   rexpr = new Variable(new TypeString(),new Iden(vName));
}
|(var = getId)
   {
              if(var == null)
                     System.out.println("VAR IS NULL!!!!");
//            rexpr.append(var.getVarName().getName());
       rexpr = var;
   }
;

getId returns [Variable v] throws CompilerException
 {
   String id = null;
   SymbolTable cur =env.getCurrentScope();
   String arrExpr = null;
   boolean isFunc = false ;
   ArrayList<Type> func = new ArrayList<Type>();
   v = null;
   ArrayList<Variable> argList = null;


 }
:str:ID ((LBRACKET arrExpr = checkArrayExpr) | (LPAREN {isFunc =
   true;} argList = checkFunc))?  // initially boolean
 {
   if(isFunc)  //  yes, it was a function cal
   {
        id = str.getText();
```

```
        // check if it is in the prototype table
        Prototype pt = env.getProto(id);
        id = CodeGen.functionCall(new
   StringBuffer(id),argList).toString();
              v = new
   Variable(Type.newInstance(pt.getReturnType()),new Iden(id));
              v.setInitialized();  // fucntions are initialized by
   default.
              argList.clear();
   } // if func!=null close
  else
  {
   id = str.getText();
   v = Variable.newInstance(cur.get(id));//bcos we change type ahead
   if(arrExpr!=null){//dealing with an array
        if(v.getVarType().getType().startsWith(Type.INT))
              v.setVarType(new TypeInt());

        if(v.getVarType().getType().startsWith(Type.STRING))
              v.setVarType(new TypeString());


        if(v.getVarType().getType().startsWith(Type.PDF))
              v.setVarType(new TypePDF());

        v.setVarName(new
   Iden(v.getVarName().getName()+"["+arrExpr+"]"));
   }// end if
  }// end else
 }

  |str2:NUMBER {v = new Variable(new TypeInt(),new
   Iden(str2.getText()));v.setInitialized();}

  |str3:STRCON {v = new Variable(new TypeString(),new
   Iden(str3.getText()));v.setInitialized();}
```

```
;

//needs to be changed for code gen
checkFunc returns[ArrayList<Variable> args]throws CompilerException
{

   Variable t = null;
// b = false;
   args = new ArrayList<Variable>();
   ArrayList<Variable> temp = new ArrayList<Variable>();
}
:RPAREN {
}
|(t=rhsOfAssign)
  {
 //    System.out.println("Temp is " + tempForFunc.size());
//  tempForFunc.add(t);
// args = new ArrayList<Type>();
// if(t == null)
//     System.out.println("in checkFunc! : got type  = NULL!!!");
   if(t!=null)
       args.add(t);

  } (temp = checkFunc)
     {
       for(int i=0;i<temp.size();i++)
              args.add(temp.get(i));
     }
 ;

//handled for code gen but needs to be tested!!
checkArrayExpr returns[String arrExp] throws CompilerException
{
// b=false;
   Variable t = null;
   arrExp = null;
```

```
}
//:RBRACKET {b=true;}
:(t = rhsOfAssign RBRACKET)
 {
   arrExp = t.getVarName().getName();
 }
// (b=checkArrayExpr)
;


printFunc throws CompilerException
{
   Variable v = null;
}
:(v = rhsOfAssign){
   out.append(CodeGen.println(new
   StringBuffer(v.getVarName().getName())));
}
;
//----------------------------Code Gen for Declarations!!-------------
   -------------------------------

declHandler throws CompilerException
{
   SymbolTable cur = env.getCurrentScope();
   int arrDim=-1;
   String token = new String();
}
:( // open lpar1
  "int"
  (
   token = strdecl arrDim = checkArray
   {
        Type t = new TypeInt();
        if(arrDim>=0){
               out.append(CodeGen.arrayDecl(new
```

```
  StringBuffer(t.getType()),new StringBuffer(token),new
  StringBuffer(Integer.toString(arrDim))));
            t = new TypeArray(t);
      }
      else{
            out.append(CodeGen.varDeclaration(new
  StringBuffer(t.getType()), new StringBuffer(token)));
      }
      out.append(CodeGen.getEndStmt());
      Variable v = new Variable(t,new Iden(token));
      if(arrDim>=0)
            v.setInitialized();
      cur.put(token,v);
  }
 )*

|"pdf"
 (
  token = strdecl arrDim = checkArray
  {
      Type t = new TypePDF();
      if(arrDim>=0){
            out.append(CodeGen.arrayDecl(new
  StringBuffer(t.getType()),new StringBuffer(token),new
  StringBuffer(Integer.toString(arrDim))));
            t = new TypeArray(t);
      }
      else{
            out.append(CodeGen.varDeclaration(new
  StringBuffer(t.getType()), new StringBuffer(token)));
      }
      out.append(CodeGen.getEndStmt());
      Variable v = new Variable(t,new Iden(token));
      if(arrDim>=0)
            v.setInitialized();
      cur.put(token,v);
```

```
   }
  )*


|"string"
 (
   token = strdecl arrDim = checkArray
   {
        Type t = new TypeString();
        if(arrDim>=0){
               out.append(CodeGen.arrayDecl(new
   StringBuffer(t.getType()),new StringBuffer(token),new
   StringBuffer(Integer.toString(arrDim))));
               t = new TypeArray(t);
        }
        else{
               out.append(CodeGen.varDeclaration(new
   StringBuffer(t.getType()), new StringBuffer(token)));
        }
        out.append(CodeGen.getEndStmt());
        Variable v = new Variable(t,new Iden(token));
        if(arrDim>=0)
               v.setInitialized();
        cur.put(token,v);
   }
  )*
 ) // close rpar1

{
   env.refreshCurrentScope(cur);
}
declHandler
| // do nothing
;

strdecl returns [String s]
```

```
{
   s=null;
}
:str2:ID {s=str2.getText();}
;

checkArray returns[int arrDim]{
   arrDim = -1;
}
:LBRACKET arrDim = checkArray
|str:NUMBER RBRACKET {arrDim = Integer.parseInt(str.getText());}
|{arrDim=-1;}//nothing
;
//----------------------------End: Code Gen for Declarations!!--------
   -------------------------------------

//--------------start Func Prototype walker--------------------------
   --
func_protos throws CompilerException
{
   Prototype p;
   Type ret;
   String name;
   ArrayList<Variable> list;

}
:#( FUNC_PROTOS
                     (ret = ret_type) (ret_type_array {     ret = new
   TypeArray(ret);     } )?
                     (name = func_name)
                     (list = args)

                     {
                            p = new Prototype(ret, name, list);
                            env.addProto(p);
                            System.out.println("Getting protos in code
```

```
   gen here!!!");
                           env.printAllProtos();
                           System.out.print("\n\n\n");
                    }
   )
   func_protos
| // null
;

ret_type_array
{

}
:LBRACKET ret_type_array
|RBRACKET

;

ret_type returns[Type t]{
   t = null;
}
:"string" {t = new TypeString();}
|"pdf" {t = new TypePDF();}
|"void" {t = new TypeVoid();}
|"int" {t = new TypeInt();}

//WHAT ABOUT RETURNING ARRAYS????!!!!?????!!!!! --> yes I am doing it
   now :)
;

func_name returns[String s]{
   s = null;
}
:(str:ID {s = str.getText();})
;
```

```
args returns[ArrayList<Variable> list]{
   Variable v;
   list = new ArrayList<Variable>();
}
:#(  ARGS (v = arg {list.add(v);})*  ) // works!!!
;

arg returns [Variable var]{
   var = null;
   Type t;
   Iden i;
}
:#(ARG (t = argType)  (i = argId) (ret_type_array {t = new
   TypeArray(t);} )? {var = new Variable(t,i);} )
;


argType returns[Type t]{
   t = null;
}
:"string" {t = new TypeString();}
|"pdf" {t = new TypePDF();}
|"int" {t = new TypeInt();}
//WHAT ABOUT RETURNING ARRAYS????!!!!?????!!!!!  - yes I am doing it.
;

argId returns[Iden i]{
   i = null;
}
:
(str:ID {i = new Iden(str.getText());})
;

//--------------end Func Prototype walker-----------------------------

func_body throws CompilerException
```

```
{
   Prototype p=null;
   Type ret=null;
   String name=null;
   ArrayList<Variable> list=null;
   SymbolTable cur =null;
   returnFlag = false;
}
:#( FUNC_BODY  {  }
   (ret = ret_type) (ret_type_array {     ret = new
   TypeArray(ret);     } )?
   (name = func_name)
   (list = args)
   {
//     p = new Prototype(ret,name,list);
       p = env.getProto(name);
//            p.setIsdefined();
//            env.refreshProto(p);

       env.addNewScope();
       cur = env.getCurrentScope();
       for(int i=0; i<list.size(); i++){
             Variable v = list.get(i);
             v.setInitialized();//this is valid bcos variables in
   func prototypes are by default initialized
             cur.put(v.getVarName().getName(),v);//adding formal
   parameters to the sym tab
       }
       env.refreshCurrentScope(cur);
       out.append(CodeGen.functionBody(new StringBuffer(new
   Variable(ret,new Iden("dummy")).getJavaType()),new
   StringBuffer(name),list));
   }

   (mainBody)
   {
```

```
        if(hasMoreCode ==true)
        {
               //throw new CompilerException("Unreachable code after
   return statement in function "+name);
        }

        if(curRetType == null && ret.getType()!=null)
        {

//            if(!ret.getType().equals(Type.VOID));
                    //throw new CompilerException("Return Statement
   for function " + name +" required");
        }
        else
        {

               // now i have the ret type of the function

               if(!ret.getType().equals(curRetType.getType()))
               {
                      System.out.println("type is
   "+curRetType.getType());
                      //throw new CompilerException("Return Type
   mismatch for function "+name);
               }

        }
   }
    RCURLY {out.append("}\n\n");hasMoreCode = false ; returnFlag =
   false;curRetType = null ; System.out.println("Printing
   all...");env.printAllScopes();env.deleteCurrentScope();}
 )
(func_body)
| // nothing
;
```

```
//body_of_func throws CompilerException
//:(mainBody)
//;
```

SpmlOut.java

Hold Java output code generated from SPML compiler

```
/**
 * @author Jayesh
 */
package spml;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.PrintStream;
import java.util.StringTokenizer;

public class SpmlOut {
   private StringBuffer out;
   public static String origFileName; // this also contains the path
   if given but not the spml extension, used for output java file
   public static String fileName; // this will store the name of the
   java class only!
   public static String outName;
   public SpmlOut(){
        out = new StringBuffer();
   }

   public void append(String str){
        out.append(str);
   }

   public void append(StringBuffer str){
```

```
        out.append(str);
    }

    public static void initFileName(String name){
        StringTokenizer st = new StringTokenizer(name,".");
        fileName = new String(st.nextToken());
        origFileName = fileName;
        int ind = fileName.lastIndexOf(File.separator);
        if(ind > -1){
               fileName = fileName.substring(ind+1);
        }
    }

    public StringBuffer getOut(){
        return out;
    }

    public void writeToJava(){
        outName = fileName+".java";
        try {
               FileOutputStream fo = new FileOutputStream(outName);
               PrintStream p = new PrintStream(fo);
               p.print(this.out.toString());
        } catch (FileNotFoundException e) {
               System.out.println("Cannot create output file");
               System.exit(0);
               //e.printStackTrace();
        }

    }
}
```

SpmlWalker.g

Perform static semantic analysis

```
// @ author : Dhivya, Jayesh
```

```
header{
   package spml;
}
{
   import java.util.*;
}


class SPMLTreeWalker extends TreeParser;
options {
   importVocab = SPML;
   defaultErrorHandler = true;
}

{
SpmlOut out = new SpmlOut();
boolean declFlag = false;
boolean controlFlag = false;
boolean returnFlag = false;
boolean hasMoreCode = false;
boolean pdfKw = false;
int lnum = 0;// this holds the line number to be used when errors are
   thrown
Type curRetType = null;
ArrayList<Type> tempForFunc = new ArrayList<Type>(1);
Env env = new Env();
}

program throws CompilerException
{
}
: #(CODE
       (func_protos)
       (mainBody) {
               // start closes here. So destroy the current scope.
```

```
//             env.deleteCurrentScope();
        }
       (func_body)
       {
              Iterator it = env.funcProtos.keySet().iterator();
              while(it.hasNext())
              {
                     String pname = (String)it.next();
                     Prototype pt = env.funcProtos.get(pname);
                     if(pt.isUsed() && !pt.getIsdefined()){
                            throw new CompilerException("Line "+lnum+":
  Function "+pt.getProtoName()+" has been used but not defined");
                     }
              }



       }
   )
   {}
;


mainBody throws CompilerException
{
   Variable dummy = null;
   String jump;
// if(returnFlag == true) // code after function return stmt.
// {
//     throw new CompilerException("Unreachable code after return
   statement");
// }

}
: #("start" {
       env.addNewScope();// adds a new Symbol Table for the current
```

```
  scope
  }
  mainBody RCURLY {System.out.println("Printing
  all...");env.printAllScopes();env.deleteCurrentScope();})
| #( DECLS {declFlag = false;}declHandler {out.append("");{declFlag =
  false; }} ) mainBody
| #( ASSIGN
  {
      String lhsId = null ;
      Type rhsType = null;
      Type lhsType = null;
      SymbolTable cur = env.getCurrentScope();
      boolean arrFlag=false;
      Variable v = null;
  }
  lhsId = strdecl (LBRACKET arrFlag = checkArrayExpr)?
  {
      v = Variable.newInstance(cur.get(lhsId)); // this is done to
  handle scoping
      if(v == null){
            throw new CompilerException("Line "+lnum+": Variable
  '"+lhsId+"' not declared");
      }
      lhsType = Type.newInstance(v.getVarType()); // this is done so
  that when I change lhsType for arrays the type in var does not get
  changed
      if(arrFlag){ // indexing has been done hence check for arrays
            if(!lhsType.getType().endsWith("]"))
                  throw new CompilerException("Line "+lnum+": Non
  array variable "+v.getVarName().getName()+" being indexed");

            if(lhsType.getType().startsWith(Type.INT))
                  lhsType = new TypeInt();
            if(lhsType.getType().startsWith(Type.PDF))
                  lhsType = new TypePDF();
            if(lhsType.getType().startsWith(Type.STRING))
```

```
                    lhsType = new TypeString();
        }
    }

    (rhsType = rhsOfAssign)
    {
              if(lhsType.getType().equals(Type.PDF) &&
    rhsType.getType().equals(Type.STRING))
              {
                       // pdf  = string is allowed. ( to open existing
    pdfs )
              }
              else
              {

                    if(!lhsType.getType().equals(rhsType.getType())){
                          throw new CompilerException("Line "+lnum+":
    Type mismatch between "+lhsType.getType()+" and
    "+rhsType.getType()+" for =");
                    }
              }
        v.setInitialized(); // since a var is to the LHS we need
    indicate that it is now initialized
        cur.replaceVar(lhsId,v); // replaces the variable if it exists
    in the current scope. If it exists in an upper scope then it adds
    it to the
                                                  // current scope. Used
    to handle scoping
        env.refreshCurrentScope(cur);

    }

    )    { }
    mainBody   // end ASSIGN

|#(OPSTMTS opstmts)mainBody
```

```
|#(CONTROLSTMT controlstmts ) mainBody
|(dummy = getId) mainBody  // this is for func calls with no lhs
|jump = jump_stmts {

if(!controlFlag)
{
   throw new CompilerException("Line "+lnum+": Statement " +jump+" is
   wrongly used here, allowed only in while block");
}

} mainBody
|return_stmt //mainBody
| //nothing
;
// this is to check that return does not come in start
return_stmt throws CompilerException
{

}

:#("return"
{
 if(!returnFlag)
 {
   throw new CompilerException("Line "+lnum+": Return Statement is
   used wrongly. Only allowed inside user defined function bodies.");
 }
}
 // ok, the function's ret type is set in the global variable.
 (curRetType = rhsOfAssign)?
 {
   if(curRetType==null)
   {
        curRetType = new TypeVoid();
   }
   else
```

```
  {System.out.println("\n Global is "+curRetType.getType());}

 })  checkformore
;

checkformore throws CompilerException
:
| { hasMoreCode = false;}
|mainBody { hasMoreCode = true; }
;


//------------control stmts (if and while) start here----------------
   ----------------------------------------------------------------

controlstmts throws CompilerException
{
   Type t=null;
   String jump;
}
:#("if"
  {
        // adding new scope
        env.addNewScope();
  }
 (t=rhsOfAssign)
{
   if(!t.getType().equals(Type.INT)){
       throw new CompilerException("Line "+lnum+": Type mismatch in if
   conditional");
   }
}
  mainBody RCURLY{System.out.println("Printing
   all...");env.printAllScopes();env.deleteCurrentScope();}(elsebody)?
   )
```

```
|#("while"
  {
        // adding new scope
        controlFlag = true;
        env.addNewScope();

  }
  t=rhsOfAssign mainBody RCURLY {controlFlag =
   false;System.out.println("Printing
   all...");env.printAllScopes();env.deleteCurrentScope();}
  )
{
   if(!t.getType().equals(Type.INT)){
       throw new CompilerException("Line "+lnum+": Type mismatch in
   while conditional");
   }
}

//|#("else" {env.addNewScope();} )
;

jump_stmts returns[String j]
{
   j = null;
}
:"continue" { j = "continue"; }
|"break" { j = "break" ; }
;

elsebody throws CompilerException
:#(ELSESTMT elsebody)
|#("else" {  env.addNewScope();} mainBody RCURLY
   {System.out.println("Printing
   all...");env.printAllScopes();env.deleteCurrentScope();})
//|//mainBody
```

```
;

//------------control stmts (if and while) end here------------------
   ---------------------------------------------------------------
opstmts throws CompilerException
{
 Variable var = null;
 Variable var2 = null;

 Type ltype = null;
 Type rtype = null;



}

:#("highlight" var = getId var2=getId)
{
   ltype = var.getVarType();
    rtype = var2.getVarType();
    if(!ltype.getType().equals(Type.PDF)
   || !rtype.getType().equals(Type.STRING)){
       throw new CompilerException("Line "+lnum+": Incompatible types
   "+ltype.getType()+","+rtype.getType()+" for highlight statement");
   }

}
|#("setauthor" var = getId var2=getId)
{
   ltype = var.getVarType();
    rtype = var2.getVarType();
    if(!ltype.getType().equals(Type.PDF)
   || !rtype.getType().equals(Type.STRING)){
       throw new CompilerException("Line "+lnum+": Incompatible types
   "+ltype.getType()+","+rtype.getType()+" for setauthor statement");
   }
}
```

```
| #("print" printFunc)

|#("totextfile" var = getId var2=getId)
{
   ltype = var.getVarType();
    rtype = var2.getVarType();
    if(!ltype.getType().equals(Type.PDF)
   || !rtype.getType().equals(Type.STRING)){
       throw new CompilerException("Line "+lnum+": Incompatible types
   "+ltype.getType()+","+rtype.getType()+" for totextfile statement");
   }
}
;


rhsOfAssign returns [Type typ] throws CompilerException
{
   typ=null;
   Variable var = null;
   Variable var2 = null;
   // constant = null;
   Type rtype = null;
   Type ltype = null;

   ArrayList<Type> funcArgTypes = null;
   //RHS can be: ID, Constant, Expression, Function, String.
    //Expression can be a combination of the above 4 types and a
   binary operator
}
:#(PLUS ltype = rhsOfAssign rtype = rhsOfAssign)
 {
//     Type ltyp = var.getVarType();
   typ = ltype.getCompatibleType(rtype, "+");
   if(typ == null)
       throw new CompilerException("Line "+lnum+": Incompatible types
   "+ltype.getType()+", "+rtype.getType()+" for operator +");
```

```
 }
|#(MINUS ltype = rhsOfAssign rtype = rhsOfAssign)
{

       typ = ltype.getCompatibleType(rtype, "-");
       if(typ == null)
              throw new CompilerException("Line "+lnum+": Incompatible
  types "+ltype.getType()+", "+rtype.getType()+" for operator -");

 }
|#(TIMES ltype = rhsOfAssign rtype = rhsOfAssign)
{
//     Type ltyp = var.getVarType();
  typ = ltype.getCompatibleType(rtype, "*");
  if(typ == null)
       throw new CompilerException("Line "+lnum+": Incompatible types
  "+ltype.getType()+", "+rtype.getType()+" for operator *");
 }
|#(DIV ltype = rhsOfAssign rtype = rhsOfAssign)
{
//     Type ltyp = var.getVarType();
  typ = ltype.getCompatibleType(rtype, "/");
  if(typ == null)
       throw new CompilerException("Line "+lnum+": Incompatible types
  "+ltype.getType()+", "+rtype.getType()+" for operator /");
 }

|#(AND ltype = rhsOfAssign rtype = rhsOfAssign)
{
  if(!ltype.getType().equals(Type.INT)||!rtype.getType().equals(Type.
  INT))
  {
       throw new CompilerException("Line "+lnum+": Incompatible types
  "+ltype.getType()+", "+rtype.getType()+" for operator &&");
  }
  typ = new TypeInt();
```

```
}

|#(OR ltype = rhsOfAssign rtype = rhsOfAssign)
{
   if(!ltype.getType().equals(Type.INT)||!rtype.getType().equals(Type.
   INT))
   {
       throw new CompilerException("Line "+lnum+": Incompatible types
   "+ltype.getType()+", "+rtype.getType()+" for operator ||");
   }
   typ = new TypeInt();
}

|#(LT ltype = rhsOfAssign rtype = rhsOfAssign)
{
   if(!ltype.getType().equals(Type.INT)||!rtype.getType().equals(Type.
   INT))
   {
       throw new CompilerException("Line "+lnum+": Incompatible types
   "+ltype.getType()+", "+rtype.getType()+" for operator <");
   }
   typ = new TypeInt();
}

|#(LEQ ltype = rhsOfAssign rtype = rhsOfAssign)
{
   if(!ltype.getType().equals(Type.INT)||!rtype.getType().equals(Type.
   INT))
   {
       throw new CompilerException("Line "+lnum+": Incompatible types
   "+ltype.getType()+", "+rtype.getType()+" for operator <=");
   }
   typ = new TypeInt();
}

|#(GT ltype = rhsOfAssign rtype = rhsOfAssign)
```

```
{
   if(!ltype.getType().equals(Type.INT)||!rtype.getType().equals(Type.
   INT))
   {
       throw new CompilerException("Line "+lnum+": Incompatible types
   "+ltype.getType()+", "+rtype.getType()+" for operator >");
   }
   typ = new TypeInt();
}

|#(GEQ ltype = rhsOfAssign rtype = rhsOfAssign)
{
   if(!ltype.getType().equals(Type.INT)||!rtype.getType().equals(Type.
   INT))
   {
       throw new CompilerException("Line "+lnum+": Incompatible types
   "+ltype.getType()+", "+rtype.getType()+" for operator >=");
   }
   typ = new TypeInt();
}


|#(EQ ltype = rhsOfAssign rtype = rhsOfAssign)
{
   if(!ltype.getType().equals(Type.INT)||!rtype.getType().equals(Type.
   INT))
   {

       if(!ltype.getType().equals(Type.PDF)||!rtype.getType().equals(T
   ype.PDF))
       {

       if(!ltype.getType().equals(Type.STRING)||!rtype.getType().equal
   s(Type.STRING))
                       throw new CompilerException("Line "+lnum+":
   Incompatible types "+ltype.getType()+", "+rtype.getType()+" for
```

```
  operator ==");
      }
  }
  typ = new TypeInt();
}

|#(INEQ ltype = rhsOfAssign rtype = rhsOfAssign)
{
  if(!ltype.getType().equals(Type.INT)||!rtype.getType().equals(Type.
  INT))
  {

      if(!ltype.getType().equals(Type.PDF)||!rtype.getType().equals(T
  ype.PDF))
      {

      if(!ltype.getType().equals(Type.STRING)||!rtype.getType().equal
  s(Type.STRING))
                  throw new CompilerException("Line "+lnum+":
  Incompatible types "+ltype.getType()+", "+rtype.getType()+" for
  operator !=");
      }

  }
  typ = new TypeInt();
}

|#("create" var = getId)
 {
  rtype = var.getVarType();
  if(!rtype.getType().equals(Type.STRING)){
      throw new CompilerException("Line "+lnum+": Incompatible types
  '"+rtype.getType()+"' for create statement");
  }
  typ = new TypePDF();
```

```
 }
|#("length" var = getId)
 {
  rtype = var.getVarType();
  if(!rtype.getType().endsWith("[]")
  && !rtype.getType().equals(Type.PDF)){
      throw new CompilerException("Line "+lnum+": Incompatible types
  '"+rtype.getType()+"' for length statement");
  }
  typ = new TypeInt();
 }
|#("getauthor" var = getId)
 {
  rtype = var.getVarType();
  if(!rtype.getType().equals(Type.PDF)){
      throw new CompilerException("Line "+lnum+": Incompatible types
  '"+rtype.getType()+"' for getauthor statement");
  }
  typ = new TypeString();
 }
|#("extractpage" var = getId var2 = getId)
 {
  ltype = var.getVarType();
  rtype = var2.getVarType();
  if(!ltype.getType().equals(Type.PDF)
  || !rtype.getType().equals(Type.INT)){
      throw new CompilerException("Line "+lnum+": Incompatible types
  '"+ltype.getType()+", "+rtype.getType()+"' for extractpage
  statement");
  }
  typ = new TypePDF();
 }
|#("in" ((var = getId)|"pdf"{pdfKw = true;}) var2 = getId)
{
  if(pdfKw == true)
  {
```

```
  rtype = var2.getVarType();
  pdfKw = false;
  ltype = new TypePDF();
  typ = ltype.getCompatibleType(rtype, "in_pdf");
  }

  else
  {
      ltype = var.getVarType();
      rtype = var2.getVarType();
      typ = ltype.getCompatibleType(rtype, "in");
  }

  if(typ == null)
      throw new CompilerException("Line "+lnum+": Incompatible types
  "+ltype.getType()+", "+rtype.getType()+" for operator in");
 }

|#(UMINUS rtype = rhsOfAssign)
{
  if(!rtype.getType().equals(Type.INT))
  {
      throw new CompilerException("Line "+lnum+": Wrong type for the
  argument of Unary Minus ~");
  }
  typ = new TypeInt();
}

|#(NOT rtype = rhsOfAssign)
{
  if(!rtype.getType().equals(Type.INT))
  {
      throw new CompilerException("Line "+lnum+": Wrong type for the
  argument of Operator NOT !");
  }
  typ = new TypeInt();
```

```
}

|(var = getId)  // todo: also check for functions and for array
   elements!!
    {
        typ = var.getVarType();
    }


;

getId returns [Variable v] throws CompilerException
 {
   String id = null;
   SymbolTable cur =env.getCurrentScope();
   boolean isItArray = false;
   boolean isFunc = false ;
   ArrayList<Type> func = new ArrayList<Type>(1);
   v = null;
   ArrayList<Type> argList = null;
 }
:str:ID ((LBRACKET isItArray = checkArrayExpr) | (LPAREN {isFunc =
   true;} argList = checkFunc))?  // initially boolean
 {

   // if it is a function -> prototype table has to be checked and not
   the
   // normal symbol table


   if(isFunc)  //  yes, it was a function cal
   {
 //          System.out.println("Got Some
   arguments!!!!!!!!!!!!!!!!!!!!!!");
//           for(int z = 0; z<argList.size(); z++)
//                  System.out.println("print:
```

```
"+argList.get(z).getType());
    id = str.getText();
    lnum = str.getLine();

    // check if it is in the prototype table
    if(env.getProto(id)==null)
    {
          throw new CompilerException("Line "+lnum+": Prototype
for the function "+id+" has not been declared before start");
    }

    // now that it has its types, check the types of its arguments
    Prototype pt = env.getProto(id);
    pt.setUsed(true);
    env.refreshProto(pt);
    ArrayList<Variable> pvars = pt.getArgList();
    if(argList == null)
          throw new CompilerException("Line "+lnum+":
Debugging!!!--> argList becomes null!!");
    if(pvars.size()!=argList.size())
    {
          throw new CompilerException("Line "+lnum+": The number
of function arguments for "+id+" do not match with function
prototype");
    }

    // some type checking to be done.
    for(int i=0;i<argList.size();i++)
    {

          //System.out.println(pvars.get(i).getVarType().getType()
+ " ");
          if(argList.get(i) == null)
                System.out.println(i+" is null!!!!!!!!!!!!!!");

    if(!argList.get(i).getType().equals(pvars.get(i).getVarType().g
```

```
 etType()))
                  {
                        throw new CompilerException("Line "+lnum+":
 The types of function call arguments for the function "+id+" do not
 match with the types in function prototype");
                  }

     }


     v = new Variable(Type.newInstance(pt.getReturnType()),new
 Iden(id));
     v.setInitialized();  // fucntions are initialized by default .
     argList.clear();
 } // if func!=null close
else
{

 id = str.getText();
 lnum = str.getLine();
 v = Variable.newInstance(cur.get(id));//bcos we change type ahead
 if(v == null)
     throw new CompilerException("Line "+lnum+": Variable '"+id+"'
 not declared");

 if(!v.isInitialized())
     throw new CompilerException("Line "+lnum+": Variable '"+id+"'
 may not have been initialized");

 if(isItArray){
     if(!v.getVarType().getType().endsWith("]"))
           throw new CompilerException("Line "+lnum+": Non array
 variable "+v.getVarName().getName()+" being indexed");
     //since var is array we change its type to be primitive of the
 array
     if(v.getVarType().getType().startsWith(Type.INT))
```

```
                v.setVarType(new TypeInt());
        else {
                if(v.getVarType().getType().startsWith(Type.STRING))
                        v.setVarType(new TypeString());

                else {
                        if(v.getVarType().getType().startsWith(Type.PDF))
                                v.setVarType(new TypePDF());
                }
        }
   }
 }
}

 |str2:NUMBER {v = new Variable(new TypeInt(),new
   Iden(str2.getText()));v.setInitialized();lnum = str2.getLine();}

 |str3:STRCON {v = new Variable(new TypeString(),new
   Iden(str3.getText())); v.setInitialized();lnum = str3.getLine();}
;

checkFunc returns[ArrayList<Type> args]throws CompilerException
{

   Type t = null;
// b = false;
   args = new ArrayList<Type>();
   ArrayList<Type> temp = new ArrayList<Type>();


}
:RPAREN {

//b = true;

    // now check for the types of the function arguments
```

```
}
|(t=rhsOfAssign)
  {
 //    System.out.println("Temp is " + tempForFunc.size());
//  tempForFunc.add(t);
// args = new ArrayList<Type>();
   if(t == null)
        System.out.println("in checkFunc! : got type  = NULL!!!");
   args.add(t);

  } (temp = checkFunc)
     {
       for(int i=0;i<temp.size();i++)
              args.add(temp.get(i));
     }
 ;

checkArrayExpr returns[boolean b] throws CompilerException
{
   b=false;
   Type t = null;
}
:RBRACKET {b=true;}
|(t = rhsOfAssign)
 {
   if(!t.getType().equals(Type.INT))
   {
        throw new CompilerException("Line "+lnum+": Expression inside
   array not int");

   }
 }
 (b=checkArrayExpr)
;
```

```
//printFunc throws CompilerException
//{
// SymbolTable cur = env.getCurrentScope();
// Variable v = null;
//}
//: str1:STRCON {} // nothing needed for stat seman checks
//| str:ID
 // {
//     String id = str.getText();
//     v = cur.get(id);
//     if(v == null)
//     throw new CompilerException("Variable '"+id+"' not declared");

// if(!v.isInitialized())
//     throw new CompilerException(" Variable '"+id+"' may not have
   been initialized");
 // }
//;
printFunc throws CompilerException
{
   Type t = null;
}
:(t = rhsOfAssign)
{
    if(!t.getType().equals(Type.INT)
   && !t.getType().equals(Type.STRING))
       throw new CompilerException("Line "+lnum+": Incompatible type
   arguments for print statement");
}

;
//----------------------------Walker for Declarations!!---------------
   ------------------------------

declHandler throws CompilerException
```

```
{
   SymbolTable cur = env.getCurrentScope();
   boolean isArray;
   String token = new String();
}
:( // open lpar1
  "int"
  (
   token = strdecl isArray = checkArray
   {
        Type t = new TypeInt();
        if(isArray)
               t = new TypeArray(t);
        Variable v = new Variable(t,new Iden(token));
        if(isArray)
               v.setInitialized();
        cur.put(token,v);
   }
  )*

|"pdf"
 (
   token = strdecl isArray = checkArray
   {
        Type t = new TypePDF();
        if(isArray)
               t = new TypeArray(t);
        Variable v = new Variable(t,new Iden(token));
        if(isArray)
               v.setInitialized();
        cur.put(token,v);
   }
  )*


|"string"
```

```
 (
   token = strdecl isArray = checkArray
   {
        Type t = new TypeString();
        if(isArray)
               t = new TypeArray(t);
        Variable v = new Variable(t,new Iden(token));
        if(isArray)
               v.setInitialized();
        cur.put(token,v);
   }
  )*
 ) // close rpar1

{
   env.refreshCurrentScope(cur);
}
declHandler
| // do nothing
;

strdecl returns [String s]
{
   s=null;
}
:str2:ID {s=str2.getText();lnum = str2.getLine();}
;

checkArray returns[boolean isArray]{
   isArray = false;
   String s;
   s = null;
// boolean temp;
}
:LBRACKET isArray = checkArray
|NUMBER isArray = checkArray
```

```
|RBRACKET {isArray = true;}
|{isArray = false;}//nothing
;
//---------------------------End: Walker for Declarations!!----------
   ----------------------------------

//--------------start Func Prototype walker--------------------------
   --
func_protos throws CompilerException
{
   Prototype p;
   Type ret;
   String name;
   ArrayList<Variable> list;

}
:#( FUNC_PROTOS
                     (ret = ret_type) (ret_type_array {     ret = new
   TypeArray(ret);     } )?
                     (name = func_name)
                     (list = args)

                     {
                            p = new Prototype(ret, name, list);
                            if(env.getProto(name)!=null)
                                   throw new CompilerException("Line
   "+lnum+": Prototype with name: "+name+" already exists");
                            env.addProto(p);
                            env.printAllProtos();
                            System.out.print("\n\n\n");
                     }
   )
   func_protos
| // null
;
```

```
ret_type_array
{

}
:LBRACKET ret_type_array
|RBRACKET

;

ret_type returns[Type t]{
   t = null;
}
:"string" {t = new TypeString();}
|"pdf" {t = new TypePDF();}
|"void" {t = new TypeVoid();}
|"int" {t = new TypeInt();}

//WHAT ABOUT RETURNING ARRAYS????!!!!?????!!!!! --> yes I am doing it
   now :)
;

func_name returns[String s]{
   s = null;
}
:(str:ID {s = str.getText();lnum = str.getLine();})
;

args returns[ArrayList<Variable> list]{
   Variable v;
   list = new ArrayList<Variable>();
}
:#(  ARGS (v = arg {list.add(v);})*  ) // works!!!
;

arg returns [Variable var]{
   var = null;
```

```
   Type t;
   Iden i;
}
:#(ARG (t = argType)  (i = argId) (ret_type_array {t = new
   TypeArray(t);} )? {var = new Variable(t,i);} )
;

argType returns[Type t]{
   t = null;
}
:"string" {t = new TypeString();}
|"pdf" {t = new TypePDF();}
|"int" {t = new TypeInt();}
//WHAT ABOUT RETURNING ARRAYS????!!!!?????!!!!!  - yes I am doing it.
;

argId returns[Iden i]{
   i = null;
}
:
(str:ID {i = new Iden(str.getText());lnum = str.getLine();})
;

//--------------end Func Prototype walker----------------------------

func_body throws CompilerException
{
   Prototype p=null;
   Type ret=null;
   String name=null;
   ArrayList<Variable> list=null;
   SymbolTable cur =null;
   returnFlag = false;
}
:#( FUNC_BODY  {  }
   (ret = ret_type) (ret_type_array {     ret = new
```

```
TypeArray(ret);     } )?
(name = func_name)
(list = args)
{
      returnFlag = true;
    p = new Prototype(ret,name,list);
    if(!env.protoExists(p))
    {
          throw new CompilerException("Line "+lnum+": Prototype
for the function '"+p.toString()+"' is not defined");
    }

    if(env.getProto(name).getIsdefined())
    {
          throw new CompilerException("Line "+lnum+": Re-
definition of function - "+p.toString());
    }
    else{
          p.setIsdefined();
          env.refreshProto(p);
    }

    env.addNewScope();
    cur = env.getCurrentScope();
    for(int i=0; i<list.size(); i++){
          Variable v = list.get(i);
          v.setInitialized();//this is valid bcos variables in
func prototypes are by default initialized
          cur.put(v.getVarName().getName(),v);//adding formal
parameters to the sym tab
    }
    env.refreshCurrentScope(cur);
}

(mainBody)
{
```

```
        if(hasMoreCode ==true)
        {
               throw new CompilerException("Line "+lnum+": Unreachable
   code after return statement in function "+name);
        }

        if(curRetType == null && ret.getType()!=null)
        {

               if(!ret.getType().equals(Type.VOID))
                      throw new CompilerException("Line "+lnum+": Return
   Statement for function " + name +" required");
        }
        else
        {

               // now i have the ret type of the function

               if(!ret.getType().equals(curRetType.getType()))
               {
                      System.out.println("type is
   "+curRetType.getType());
                      throw new CompilerException("Line "+lnum+": Return
   Type mismatch for function "+name);
               }

        }
   }
    RCURLY {hasMoreCode = false ; returnFlag = false;curRetType =
   null ; System.out.println("Printing
   all...");env.printAllScopes();env.deleteCurrentScope();}
 )
(func_body)
| // nothing
;
```

```
//body_of_func throws CompilerException
//:(mainBody)
//;
```

SymbolTable.java

Represent a symbol table

```
package spml;

import java.util.Enumeration;
import java.util.Hashtable;

/**
 * @author Jayesh and Dhivya
 */
public class SymbolTable {
   private Hashtable<String, Variable> table;
   protected SymbolTable outer; // ST for enclosing scope

   public SymbolTable(SymbolTable outer) {
       table = new Hashtable<String, Variable>();
       this.outer = outer;
   }

   public SymbolTable(){

   }

   public void put(String token, Variable v) throws CompilerException{
       if(get(token)!=null){ //checks if same var exists in outer
   scopes bcos not allowed in java
               throw new CompilerException("Variable " +token+" already
   declared");
       }
       this.table.put(token,v);
   }
```

```
  public void replaceVar(String token, Variable v){
      if(this.table.containsKey(token))
            this.table.remove(token);

      this.table.put(token, v);
  }
//  Search in this and outer scopes for an identifier
  public Variable get(String token) {
  for (SymbolTable tab = this ; tab != null ; tab = tab.outer) {
      Variable id = tab.table.get(token);
      if ( id != null ) return id;
  }
  return null;
  }

  public Variable getFromCurrent(String token){
      return this.table.get(token);
  }

  public void printTable(){
      Enumeration<String> en = table.keys();
      while(en.hasMoreElements()){
            String str = en.nextElement();
            Variable v = table.get(str);
            System.out.println(v.toString());
      }
  }
}
```

Type.java

Abstract class for types in SPML

```
package spml;
```

```
/** Type class represent the type of a token during the compilation.
 * @author stefy
 *
 */
public abstract class Type {

   /**
    * INT represents the string used in SPML syntax for intger type
    */
   public static final String INT = "int";

   /**
    * STRING represents the string used in SPML syntax for string type
    */
   public static final String STRING = "string";

   /**
    * PDF represent the string used in SPML syntax for pdf type
    */
   public static final String PDF = "pdf";

   /**
    * VOID represent the string used in SPML syntax for void type
    */
   public static final String VOID = "void";

   public static final String JAVA_INT = "int";
   public static final String JAVA_STRING = "String";
   public static final String JAVA_PDF= "PDFItem";
   public static final String JAVA_VOID = "void";

   protected String type;


   public String getType(){
       return this.type;
```

```
}

public static Type newInstance(Type t){
    Type result = null;
    String type = t.getType();
    if(type.startsWith(Type.INT))
          result = new TypeInt();

    if(type.startsWith(Type.PDF))
          result = new TypePDF();

    if(type.startsWith(Type.VOID)){
          result = new TypeVoid();
          return result;//bcos if type is void then arrays are not
possible
    }
    if(type.startsWith(Type.STRING))
          result = new TypeString();

    if(type.endsWith("[]"))
          result = new TypeArray(result);

    return result;
}



/** Rule for implicit type casting. This phenomenon happens only
when adding integers to strings or viceversa.
 * All the other combinations will throw a Compiler Exception
 * @param t1 One of the two Type to be possibly converted
 * @param t2 The other one
 * @return The implicit cast type
 * @throws CompilerException
 */
```

```
public Type implicitCast(Type t1, Type t2) throws
CompilerException{
    if(
           (t1.getType() == Type.STRING && t2.getType()== Type.INT)
||
           (t1.getType() == Type.INT && t2.getType() ==
Type.STRING)
           )
           return new TypeString();
    else
           throw new CompilerException("Invalid implicit type
cast");
}


public Type getCompatibleType(Type rtype, String op){
    String ltyp = this.getType();
    String rtyp = rtype.getType();
    if(op.equals("+")){
           if(ltyp.equals(Type.INT)){
                  if(rtyp.equals(Type.INT))
                         return new TypeInt();
                  if(rtyp.equals(Type.STRING))
                         return new TypeString();
           }
           if(ltyp.equals(Type.STRING)){
                  if(rtyp.equals(Type.INT))
                         return new TypeString();
                  if(rtyp.equals(Type.STRING))
                         return new TypeString();
           }
           if(ltyp.equals(Type.PDF)){
                  if(rtyp.equals(Type.PDF))
                         return new TypePDF();

           }
```

```
        }

        if(op.equals("-")||op.equals("*")||op.equals("/")){
               if(ltyp.equals(Type.INT)){
                      if(rtyp.equals(Type.INT))
                             return new TypeInt();
               }
        }

        if(op.equals("in")){
               if(ltyp.equals(Type.STRING)){
                      if(rtyp.equals(Type.STRING))
                             return new TypeInt();

                      if(rtyp.equals(Type.PDF)){
                             Type t = new TypeInt();
                             return new TypeArray(t);
                      }
               }

               if(ltyp.equals(Type.PDF)){
                      if(rtyp.equals(Type.STRING))
                             return new TypeInt();
               }
        }

        if(op.equals("in_pdf")){
               if(rtyp.equals(Type.STRING)){
                      Type t = new TypePDF();
                      return new TypeArray(t);
               }
        }

        return null; // incompatible types!!!!!
    }
}
```

TypeArray.java

Represent an array for types available in SPML (int, string, and pdf)

```
package spml;

/** Array is a subclass of Type, meant to indentify array types
 * @author stefy
 *
 */
public class TypeArray extends Type {
   public static final String TYPE_AR_INT = "int[]";
   public static final String TYPE_AR_STRING = "string[]";
   public static final String TYPE_AR_PDF = "pdf[]";
   public TypeArray(Type t) {
        this.type = t.getType()+"[]";
   }
}
```

TypeInt.java

Represent the int type

```
/*
 * @ author : stefy
 */

package spml;

public class TypeInt extends Type {
   public TypeInt(){
        this.type = Type.INT;
   }

}
```

TypePDF.java

Represent the pdf type

```
/*
 * @ author : stefy
 */

package spml;

public class TypePDF extends Type{
   public TypePDF(){
        this.type = Type.PDF;
   }
}
```

TypeString.java

Represent the string type

```
/*
 * @ author : stefy
 */

package spml;

public class TypeString extends Type{
   public TypeString(){
        this.type = Type.STRING;
   }
}
```

TypeVoid.java

Represent the void type

```
/**
 *
 */
package spml;
```

```
/**
 * @author JayeshKataria
 *
 */
public class TypeVoid extends Type{
   public TypeVoid(){
        this.type = Type.VOID;
   }
}
```

Variable.java

Represent a variable

```
package spml;

/**
 * This class is an encapsulation for a variable in SPML.
 * It contains for every variable, its type and its name
 * author : Jayesh and Stefy
 */

public class Variable {
   private Type varType;
   private Iden varName;
   private boolean valPresent;//checks if there is a valid value to it
   before it is used anywhere
   //needed bcos java will give an error otherwise

   public Variable(Type t, Iden i){
        varType = t;
        varName = i;
        valPresent = false;
   }

   public static Variable newInstance(Variable v){
```

```
    if(v == null)
           return null;

   Variable result = new Variable(Type.newInstance(v.getVarType()),
Iden.newInstance(v.getVarName()));
    if(v.isInitialized())
           result.setInitialized();
    return result;
}
public Iden getVarName() {
    return varName;
}

public void setVarName(Iden var_name) {
    this.varName = var_name;
}

public Type getVarType() {
    return varType;
}

public String getJavaType() throws CompilerException{
    Type temp = this.varType;
    boolean arrFlag = false;

    if(this.varType.getType().endsWith("[]")){
           arrFlag = true;
           if(this.varType.getType().startsWith(Type.PDF))
                  temp = new TypePDF();

           if(this.varType.getType().startsWith(Type.STRING))
                  temp = new TypeString();

           if(this.varType.getType().startsWith(Type.INT))
                  temp = new TypeInt();
    }
```

```
        if(temp instanceof TypePDF){
               if(arrFlag)
                      return Type.JAVA_PDF+"[]";
               else
                      return Type.JAVA_PDF;
        }

        if(temp instanceof TypeInt){
               if(arrFlag)
                      return Type.JAVA_INT+"[]";
               else
                      return Type.JAVA_INT;
        }

        if(temp instanceof TypeString){
               if(arrFlag)
                      return Type.JAVA_STRING+"[]";
               else
                      return Type.JAVA_STRING;
        }

        if(temp instanceof TypeVoid)
               return Type.JAVA_VOID;
//      if(this.varType instanceof TypeArray)
//             return this.varType.getType();
        throw new CompilerException("SpmlException: Non Existent
    Type");

    }

    public void setVarType(Type var_type) {
        this.varType = var_type;
    }

    public boolean isInitialized(){
        return valPresent;
```

```
    }

    public void setInitialized(){
        valPresent = true;
    }

    public String toString(){
        return varType.getType() + " " + varName.getName() + " Init:" +
    this.isInitialized();
    }

}
```

Code Generation

CodeGen.java

Generate Java output code line by line

```
package spml;

import java.util.ArrayList;
import java.util.Random;



/**THis class provides the code generation methods.
 *
 * @author Stefano Pacifico and Jayesh
 *
 */
public class CodeGen {
    private static String file_name;
    public final static String TMPPREFIX = "SPMLTMP";

    public static void setFilename(String s){
        file_name = s;
```

```
}

public static String randomNameGen(String s){
    Random r = new Random();
    double result = Math.floor(r.nextDouble()*100000)/100000;
    return s+Double.toString(result).substring(2);
}

public static String randomTmpFileNameGen(String s){
    Random r = new Random();
    double result = Math.floor(r.nextDouble()*100000)/100000;
    return s+Double.toString(result).substring(2)+".pdf";
}

public static String getFilename(){
    return file_name;
}


/**Method to write a left parenthsis in the code.
 * @return a StringBuffer containing left parenthsis character
 */
public static StringBuffer getLParen(){
    return new StringBuffer("(");
}

/**Method to write a left parenthsis in the code.
 * @return a StringBuffer containing the right parenthsis character
 */
public static StringBuffer getRParen(){
    return new StringBuffer(")");
}

/**Method to write a left curly bracket in the code.
 * @returna a StringBuffer containing the left curly bracket
character
```

```
 */
public static StringBuffer getLCurly(){
    return new StringBuffer("{");
}

/**Method to write a right curly bracket in the code.
 * @return a StringBuffer containing the right bracket character
 */
public static StringBuffer getRCurly(){
    return new StringBuffer("}");
}

/**Method to write a left bracket in the code.
 * @return a StringBuffer containing the left bracket character
 */
public static StringBuffer getLBracket(){
    return new StringBuffer("[");
}

/**Method to write a right bracket in the code.
 * @return a StringBuffer containing the right bracket character
 */
public static StringBuffer getRBracket(){
    return new StringBuffer("]");
}

/** This method adds a semicolon and a newline in the code. Useful
to complete the statements.
 * @return a StringBuffer containing ";\n" string.
 */
public static StringBuffer getEndStmt(){
    return new StringBuffer(";\n");
}

/**Generate the code for the while keyword
 * @return a StringBuffer containing the java 'while' keyword
```

```
 */
public static StringBuffer whileStmt(StringBuffer expr){
    return new StringBuffer("while("+expr+") {\n");
}

/**Generate the cod for the if keyword
 * @return  a StringBuffer containing the java 'while' keyword
 */
public static StringBuffer ifStmt(StringBuffer expr){
    return new StringBuffer("if("+expr+") {\n");
}

/**Generate the code to retrieve the length of an array
 * @param p the array variable name
 * @return
 */
public static StringBuffer arrayLength(StringBuffer varname) {
    StringBuffer foo = new StringBuffer();
    return foo.append(varname+".length");
}

public static StringBuffer print(StringBuffer p){
    StringBuffer foo = new StringBuffer();
    return
foo.append("System.out.print(").append(p).append(");\n");
}

public static StringBuffer println(StringBuffer p){
    StringBuffer foo = new StringBuffer();
    return
foo.append("System.out.println(").append(p).append(");\n");
}

public static StringBuffer pdfInDirStmt(StringBuffer lexpr,
StringBuffer rexpr) {
    return new
```

```
StringBuffer("SPMLLibrary.listPDFFiles("+rexpr+".toString())");
}

public static StringBuffer assignment(StringBuffer lexpr,
StringBuffer rexpr) {
    StringBuffer foo = new StringBuffer();
    foo.append(lexpr);
    foo.append("=");
    foo.append(rexpr);
    return foo;
}

public static StringBuffer pdfCreate(StringBuffer expr){
    StringBuffer foo = new StringBuffer();
//  foo.append("PDFItem ");
//  foo.append(rexpr);
//  foo.append(" = ");
    foo.append("new PDFItem("+expr+")");
    return foo;
}

/*public static StringBuffer pdfDeclaration(StringBuffer expr){
    StringBuffer foo = new StringBuffer("PDFItem "+expr);
    return foo;
}*/

public static StringBuffer pdfAssignment(StringBuffer lexpr,
StringBuffer rexpr){
    StringBuffer foo = new StringBuffer();
    foo.append("if("+lexpr+"== null)\n");
    foo.append(lexpr+"="+rexpr+";");
    foo.append("else {\n");
    String tmpvarname = "_tmp";
    foo.append("File "+tmpvarname+"= new
File(CodeGen.TMPPREFIX+\"secret.pdf\");\n");
    foo.append("FileOutputStream _o = new
```

```
FileOutputStream("+tmpvarname+");\n" +
                        "FileInputStream _i = new
FileInputStream(new File("+rexpr+".getPath()));\n" +
                        "byte[] buf = new byte[1024];\n" +
                        "int len;\n" +
                        "while ((len = _i.read(buf)) > 0) {\n" +
                        "_o.write(buf, 0, len);\n" +
                        "}\n" +
                        "_i.close();\n" +
                        "_o.close();\n" +
                        lexpr+".getPdfFile().delete();\n"+
                        tmpvarname+".renameTo("+
                        lexpr+".getPdfFile() );\n"+
                        "}\n"
    );
    return foo;
}

/**Generates the code to access an existing PDF file
 * @param lexpr is the name of the variable holding the reference
to the PDF file
 * @param rexpr is the pdf file path.
 * @return the stringbuffer containing the code to be added.
 */
public static StringBuffer pdfOpen(StringBuffer lexpr, StringBuffer
rexpr){
    StringBuffer foo = new StringBuffer();
    String file_var = randomNameGen("file");
    foo.append("File "+file_var+" = new File("+rexpr+");\n");
    foo.append("if(!"+file_var+".exists())\n");
    foo.append("throw new RuntimeException(\"PDF file doesn't
exist; cannot open\");\n");
    foo.append(lexpr+"=");
    foo.append(pdfCreate(rexpr));
    return foo;
```

```
}

/**Generate the code for a variable declaration
 * @param type The spml type of the variable
 * @param varname The spml name of the variable
 * @return The java declaration statement
 */
public static StringBuffer varDeclaration(StringBuffer type,
StringBuffer varname){
    StringBuffer foo = new StringBuffer();
    if(type.toString().equals(Type.INT))
          foo.append("int "+varname.toString());
    if(type.toString().equals(Type.STRING))
          foo.append("String "+varname.toString());
    if(type.toString().equals(Type.PDF))
          foo.append("PDFItem "+varname.toString()+"= null");
    return foo;
}

/**Generates the code for an array declaration
 * @param type the type of the array variable
 * @param varname the name of the variable being declarated
 * @param array_dim the dimension of the array
 * @return
 */
public static StringBuffer arrayDecl(StringBuffer type,
StringBuffer varname, StringBuffer array_dim){
    StringBuffer foo = new StringBuffer();
    if(type.toString().equals(Type.INT)) {
          foo.append(Type.JAVA_INT+"[] "+varname.toString());
          foo.append(" = ");
          foo.append("new "+Type.JAVA_INT+"["+array_dim+"]");
    }
    if(type.toString().equals(Type.STRING)){
          foo.append(Type.JAVA_STRING + "[] "+varname.toString());
          foo.append(" = ");
```

```
            foo.append("new "+Type.JAVA_STRING+"["+array_dim+"]");
        }

        if(type.toString().equals(Type.PDF)){
            foo.append(Type.JAVA_PDF+ "[] "+varname.toString());
            foo.append(" = ");
            foo.append("new "+Type.JAVA_PDF+"["+array_dim+"];\n");
            foo.append("for(int i = 0; i<"+varname+".length;
i++)\n");
            foo.append(varname+"[i] = null;\n");
        }

        // Last semicolon should be put by the walker
        return foo;
    }

    /**Generate the code for the 'totext' function.
     * @param lexpr is the StringBuffer containing the name of the pdf
    variable referring to the PDF document to read
     * @param rexpr is the StringBuffer containing the path of the text
    file where to write into.
     * @return the StringBuffer containing the code generated
     */
    public static StringBuffer totextStmt(StringBuffer lexpr,
    StringBuffer rexpr){
        StringBuffer foo = new StringBuffer();
        //Create the names of the variables
        String ip_var = randomNameGen("ip");
        String pdfdoc_var =randomNameGen("pdfdoc");
        String word_list_var =randomNameGen("word_list");
        String li_var = randomNameGen("li");
        String word_var = randomNameGen("word");
        String op_var =  randomNameGen("op");

        foo.append("InputStream "+ip_var+";\n");
        foo.append("FileWriter "+op_var+";\n");
```

```
    foo.append("PDFDocument "+pdfdoc_var+";\n");
    foo.append("ArrayList<PDFWord> "+word_list_var+" = new
ArrayList<PDFWord>();\n");

    foo.append(op_var+" = new FileWriter(new File("+rexpr+"));\n");
    foo.append(ip_var+" = new
FileInputStream("+lexpr+".getPath());\n");
    foo.append(pdfdoc_var+" =
PDFFactory.openDocument("+ip_var+");\n");

    foo.append("ListIterator "+li_var+" =
"+pdfdoc_var+".getWords();\n");

     foo.append("while("+li_var+".hasNext()){\n");
     foo.append("PDFWord "+word_var+" =
(PDFWord)"+li_var+".next();\n");
     foo.append(op_var+".write("+word_var+".getString() +\"
\" );\n");
     foo.append("}\n");
     foo.append(op_var+".close()\n");
     return foo;
}

/** Generates the code for concatenating PDF files with the plus
operator
 * @param lexpr the name of the first pdf variable
 * @param rexpr the name of the second pdf variable
 * @return the StringBuffer holding the entire code generated
 */
public static StringBuffer pdfSum(StringBuffer lexpr, StringBuffer
rexpr){
    StringBuffer foo = new StringBuffer();
    foo.append(lexpr+".sum("+rexpr+")");
    return foo;
}
```

```
/** Generates the code for string in string statement
 * @param lexpr is the stringbuffer holding the lefthand string
expression
 * @param rexpr is the stringbuffer holding the righthand string
expression
 * @param variable is the variable holding the result of the
evaluation
 * @return the StringBuffer holding the entire code generated
 */
public static StringBuffer strInStrStmt(StringBuffer lexpr,
StringBuffer rexpr, String variable){
    StringBuffer foo = new StringBuffer();

    String str1 = randomNameGen("str1");
    String str2 = randomNameGen("str2");

    foo.append("int "+variable+";\n");
    foo.append("String " + str1 + " = "+lexpr+";\n");
    foo.append("String " + str2 + " = "+rexpr+";\n");
    //?????
    foo.append("if("+str2+".indexOf("+str1+") >= 0 )\n ");
    foo.append(variable+"= 1;\n");
    foo.append("else\n");

    // Last semicolon should be put by the walker
    foo.append(variable+"= 0\n");
    return foo;
}

/**Generate the code for the highilight pdf operation
 * @param lexpr The StringBuffer containing name of the variable
holding the reference to the pdf file
 * @param rexpr The StringBuffer containing the string to look for
in the pdf file
 * @return the StringBuffer the code to be printed in the
outputfile
```

```
 */
public static StringBuffer highlight(StringBuffer lexpr,
StringBuffer rexpr){
    StringBuffer foo = new StringBuffer();
    foo.append("SPMLLibrary.highlightWord("+lexpr+".getPath(),"+
rexpr +")");
    return foo;
}

/* Generate the code for
 * @param array_type
 * @param array_name
 * @return
 *
public static StringBuffer arrayDeclWithoutDim(StringBuffer
array_type, StringBuffer array_name){
    StringBuffer foo = new StringBuffer();
    foo.append(array_type+"[] ");
    foo.append(array_name);
    return foo;
}*/

/**Generate the code for the statement 'string in pdf' which calls
the appropriate function from the library
 * @param lexpr is the StringBuffer containing the expression
representing the string to be searched on the pdf file
 * @param rexpr is the StringBuffer containing the expression
representing the pdf file where to search in
 * @return the StringBuffer containing the code generated
 */
public static StringBuffer stringInPdfStmt(StringBuffer lexpr,
StringBuffer rexpr){
    StringBuffer foo = new StringBuffer();
    foo.append("SPMLLibrary.stringInPdf(");
    foo.append(lexpr);
    foo.append(",");
```

```
    foo.append(rexpr);
    foo.append(")");
    return foo;
}

/**Generate the code to create the main method in the output code.
 * @return the StringBuffer containing the code generated
 */
public static StringBuffer mainStmt(){
    StringBuffer foo = new StringBuffer();
    foo.append("public static void main(String[] args) throws
Exception {\n");
    return foo;
}


/**Generate the code for arithmetic expressions for intgers and
strings
 * @param lexpr is the StringBuffer containing the expression of
the left operand
 * @param rexpr is the StringBuffer containing the expression of
the right operand
 * @param operator is the StringBuffer containing the operator used
 * @return the StringBuffer containing the code gnerated
 */
public static StringBuffer expressionGen(StringBuffer lexpr,
StringBuffer rexpr, StringBuffer operator){
    StringBuffer foo = new StringBuffer();
/*  if(operator.toString().equals("&&")){
          foo.append("( ("+lexpr+"!= 0)&&("+rexpr+" != 0) )");
          return foo;
    }
    if(operator.toString().equals("||")){
          foo.append("( ("+lexpr+"!= 0)||("+rexpr+" != 0) )");
          return foo;
    }*/
```

```
    foo.append(lexpr);
    foo.append(operator);
    foo.append(rexpr);
    return foo;
}
/**Generate the code for the statement 'extract' which calls the
appropriate function from the library
 * @param lexpr is the StringBuffer containing the name of the pdf
file where to extract the page from
 * @param rexpr is the StringBuffer containing the expression
representing the pdf file where to search in
 * @return the StringBuffer containing the code generated
 */
public static StringBuffer extractStmt(StringBuffer lexpr,
StringBuffer rexpr){
    StringBuffer foo = new StringBuffer();

    foo.append("SPMLLibrary.extractPage("+lexpr+".getPath(),"+rexpr
+")");
    return foo;
}

/**Generate the code to put the unary munus in front of expressions
 * @param expr is the StringBuffer containing the expression to
complement
 * @return the StringBuffer contating the code generated
 */
public static StringBuffer unaryMinus(StringBuffer expr){
    StringBuffer foo = new StringBuffer();
    foo.append("-").append(expr);
    return foo;
}

/**Generate the code to evaluate an intger expression as a java
boolean one
 * @param expr is the StringBuffer containing the expression to
```

```
evaluated
 * @return the StringBuffer contating the code generated
 */
public static StringBuffer unaryNotBool(StringBuffer expr){
    StringBuffer foo = new StringBuffer();
    foo.append("("+expr+"==0)");
    return foo;
}
/**Generate the code to evaluate a boolean java expression as a
integer one(0 = false, else true)
 * @param expr is the StringBuffer containing the expression to
evaluated
 * @return the StringBuffer contating the code generated
 */
public static StringBuffer unaryNotInt(StringBuffer expr){
    StringBuffer foo = new StringBuffer();
    foo.append("("+expr+"!=0)?0:1");
    return foo;
}

/**Generate the code for the else statement
 * @return the StringBuffer contating the code generated
 */
public static StringBuffer elseStmt(){
    StringBuffer foo = new StringBuffer();
    foo.append("else{\n");
    return foo;
}

/**Generate the code for a function call.
 * @param name is the StringBuffer containing the name of the
function to be called
 * @param list is the ArrayList of Variable type elements,
containing the actual parameters to be passed
 * @return the StringBuffer contating the code generated
 */
```

```
public static StringBuffer functionCall(StringBuffer name,
ArrayList<Variable> list){
    StringBuffer foo = new StringBuffer();
    foo.append(name+"(");
    for (int i =0; i< list.size(); i++){
           Variable v = list.get(i);
           foo.append(v.getVarName().getName());
           if(i!=list.size()-1)
                  foo.append(",");
    }
    foo.append(")");
    return foo;
}

/**Generate the code to create the java method corresponding to the
function definition
 * @param retType is the StringBuffer containing the JAVA retrun
type of the method
 * @param name is the StringBuffer containing the function name
 * @param list list is the ArrayList of Variable type elements,
containing the actual parameters to be passed
 * @return the StringBuffer containing the code generated
 * @throws CompilerException
 */
public static StringBuffer functionBody(StringBuffer retType,
StringBuffer name, ArrayList<Variable> list) throws
CompilerException{
    StringBuffer foo = new StringBuffer();
    foo.append("static "+retType+" "+name+"(");
    for (int i =0; i< list.size(); i++){
           Variable v = list.get(i);
           foo.append(v.getJavaType()+" ");
           foo.append(v.getVarName().getName());
           if(i!=list.size()-1)
                  foo.append(",");
    }
```

```
    foo.append(") throws Exception{\n");
    return foo;
}

/** Generate the code for the logical equality operator applied to
pdf elements
 * @param lexpr is the StringBuffer containing the name of the
lefthand pdf variable
 * @param rexpr is the StringBuffer containing the name of the
righthand pdf variable
 * @return the StringBuffer containing the code generated
 */
public static StringBuffer pdfEqualityInt(StringBuffer lexpr,
StringBuffer rexpr){
    StringBuffer foo = new StringBuffer();
    foo.append("("+lexpr+".getPath().equals(");
    foo.append(rexpr+".getPath() )");
    foo.append(")?1:0");
    return foo;
}


/** Generate the code for the logical equality operator applied to
string expressions
 * @param lexpr is the StringBuffer containing the name of the
lefthand string expression
 * @param rexpr is the StringBuffer containing the name of the
righthand string expression
 * @return the StringBuffer containing the code generated
 */
public static StringBuffer stringEqualityInt(StringBuffer lexpr,
StringBuffer rexpr){
    StringBuffer foo = new StringBuffer();
    foo.append("("+lexpr+".equals(");
    foo.append(rexpr+" )");
    foo.append(")?1:0");
```

```
    return foo;
}


/** Generate the code for the logical inequality operator applied
to string expressions
 * @param lexpr is the StringBuffer containing the name of the
lefthand pdf variable
 * @param rexpr is the StringBuffer containing the name of the
righthand pdf expression
 * @return the StringBuffer containing the code generated
 */
public static StringBuffer pdfInequalityInt(StringBuffer lexpr,
StringBuffer rexpr){
    StringBuffer foo = new StringBuffer();
    foo.append("("+lexpr+".getPath().equals(");
    foo.append(rexpr+".getPath() )");
    foo.append(")?0:1");
    return foo;
}

/** Generate the code for logical inequality between strings  in
JAVA eventually evaluated as integers
 * @param lexpr is the expression representing the left operand
 * @param rexpr is the expression for representing the right
operand
 * @return the StringBuffer containing the code generated
 */
public static StringBuffer stringInequalityInt(StringBuffer lexpr,
StringBuffer rexpr){
    StringBuffer foo = new StringBuffer();
    foo.append("("+lexpr+".equals(");
    foo.append(rexpr+" )");
    foo.append(")?0:1");
    return foo;
}
```

```
/** Generate the code for the logical equality between pdf
variables eventually evaluated as integer
 * @param lexpr lexpr is the expression representing the left
operand
 * @param rexpr is the expression for representing the right
operand
 * @return the StringBuffer containing the code generated
 */
public static StringBuffer pdfEqualityBool(StringBuffer lexpr,
StringBuffer rexpr){
    StringBuffer foo = new StringBuffer();
    foo.append(lexpr+".getPath().equals(");
    foo.append(rexpr+".getPath() )");
    return foo;
}

/** Generate the code for the logical equality between strings to
be evaluated in a JAVA boolean expression
 * @param lexpr lexpr is the expression representing the left
operand
 * @param rexpr lexpr is the expression representing the right
operand
 * @return the StringBuffer containing the code generated
 */
public static StringBuffer stringEqualityBool(StringBuffer lexpr,
StringBuffer rexpr){
    StringBuffer foo = new StringBuffer();
    foo.append(lexpr+".equals(");
    foo.append(rexpr+" )");
    return foo;
}

/**Generate the the code for the lof
 * @param lexpr
 * @param rexpr
```

```
 * @return the StringBuffer containing the code generated
 */
public static StringBuffer pdfInequalityBool(StringBuffer lexpr,
StringBuffer rexpr){
    StringBuffer foo = new StringBuffer();
    foo.append("!"+lexpr+".getPath().equals(");
    foo.append(rexpr+".getPath())");
    return foo;
}

/**
 * @param lexpr
 * @param rexpr
 * @return the StringBuffer containing the code generated
 */
public static StringBuffer stringInequalityBool(StringBuffer lexpr,
StringBuffer rexpr){
    StringBuffer foo = new StringBuffer();
    foo.append("!"+lexpr+".equals(");
    foo.append(rexpr+" )");
    return foo;
}

/**
 * @param expr
 * @return the StringBuffer containing the code generated
 */
public static StringBuffer pdfLength(StringBuffer expr){
    StringBuffer foo = new StringBuffer();
    foo.append("SPMLLibrary.pageNum(");
    foo.append(expr+".getPath()");
    foo.append(")");
    return foo;
}

/**
```

```
 * @param lexpr
 * @param rexpr
 * @param operator
 * @return the StringBuffer containing the code generated
 */
public static StringBuffer expressionGenBoolToInt(StringBuffer
lexpr, StringBuffer rexpr, StringBuffer operator){
    StringBuffer foo = new StringBuffer();
    if(operator.toString().equals("&&")){
           foo.append("( ("+lexpr+"!= 0)&&("+rexpr+" != 0) )?1:0");
           return foo;
    }
    if(operator.toString().equals("||")){
           foo.append("( ("+lexpr+"!= 0)||("+rexpr+" != 0) )?1:0");
           return foo;
    }
    foo.append("("+lexpr);
    foo.append(operator);
    foo.append(rexpr+")");
    foo.append("?1:0");
    return foo;
}
/**
 * @param lexpr
 * @param rexpr
 * @param operator
 * @return the StringBuffer containing the code generated
 */
public static StringBuffer expressionGenIntToBool(StringBuffer
lexpr, StringBuffer rexpr, StringBuffer operator){
    StringBuffer foo = new StringBuffer();
    foo.append("( ("+lexpr+operator+rexpr+") != 0 )");
    return foo;
}

/**
```

```
 * @param lexpr
 * @param rexpr
 * @return the StringBuffer containing the code generated
 */
public static StringBuffer pdfExistInDir(StringBuffer lexpr,
StringBuffer rexpr){
    StringBuffer foo = new StringBuffer();
    foo.append("(new
File("+rexpr+"+"+lexpr+".getPath().substring("+lexpr+".getPath().la
stIndexOf(File.pathSeparator)+1))).exists()?1:0");
    return foo;
}

/**
 * @return the StringBuffer containing the code generated
 */
public static StringBuffer deleteTmpFileGen(){
    StringBuffer foo = new StringBuffer();
    foo.append("SPMLLibrary.deleteTmpFiles()");
    return foo;
}

/**
 * @param expr
 * @return the StringBuffer containing the code generated
 */
public static StringBuffer intExprEval(StringBuffer expr){
    if (!isBooleanExpr(expr) ) {
           StringBuffer foo = new StringBuffer();
           foo.append("("+expr+") != 0");
           return foo;
    }
    return expr;
}
/**
 * @param expr
```

```
     * @return the StringBuffer containing the code generated
     */
    private static boolean isBooleanExpr(StringBuffer expr){
        if (
                        (expr.toString().lastIndexOf("<") == -1)
                      &&(expr.toString().lastIndexOf("<=") == -1)
                      &&(expr.toString().lastIndexOf(">") == -1)
                      &&(expr.toString().lastIndexOf(">=") == -1)
                      &&(expr.toString().lastIndexOf("&&") == -1)
                      &&(expr.toString().lastIndexOf("||") == -1)
                      &&(expr.toString().lastIndexOf("!") == -1)
                      &&(expr.toString().lastIndexOf("==") == -1)
              ) return false;
        return true;
    }
}
```

SPMLLibrary.java

Bridge class between Java output code and external libraries (iText and XPAAJ)

```
package spml;

import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FilenameFilter;
import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.ListIterator;

import com.adobe.pdf.*;
```

```
import com.lowagie.text.Document;
import com.lowagie.text.DocumentException;
import com.lowagie.text.pdf.PRAcroForm;
import com.lowagie.text.pdf.PdfContentByte;
import com.lowagie.text.pdf.PdfCopy;
import com.lowagie.text.pdf.PdfImportedPage;
import com.lowagie.text.pdf.PdfReader;
import com.lowagie.text.pdf.PdfStamper;
import com.lowagie.text.pdf.PdfWriter;
import com.lowagie.text.pdf.SimpleBookmark;
import com.lowagie.text.xml.xmp.XmpWriter;

import spml.*;

/**
 * PDFUtil provides a collection of helpers and accessory functions
   to use in code generation
 * @author stefy
 *
 */
public  class SPMLLibrary {
   /**
    *
    * @param path is the directory path where to extract the pdf files
   from
    * @return String[] array containing all the PDF files contained in
   the directory specified
    */

    public static PDFItem[] listPDFFiles(String path){
        File f = new File(path);
        MyPDFFileFilter filter = new MyPDFFileFilter();
        String[] list = f.list(filter);
        PDFItem p[] = new PDFItem[list.length];
        for(int i = 0; i<list.length; i++)
             p[i] = new PDFItem(path+File.separator+list[i]);
```

```
      return p;
  }


public static PDFItem extractPage(String file_path,  int
page_number) throws Exception, IOException, DocumentException{
    if(page_number < 1 || page_number > pageNum(file_path))
          throw new Exception("SPML Runtime Exception: trying to
access a page non existent for "+file_path);
    Document document = null;
     PdfCopy  writer = null;

     //String tmpfile =
"tmpfile"+Long.toString(System.currentTimeMillis());
     // we create a reader for the first file

     PDFItem p = new PDFItem();

     PdfReader reader = new PdfReader(file_path);
     reader.consolidateNamedDestinations();

     // step 1: creation of a document-object
     document = new Document(reader.getPageSizeWithRotation(1));
     // step 2: we create a writer that listens to the document
     writer = new PdfCopy(document, new
FileOutputStream(p.getPath()));
     // step 3: we open the document
     document.open();

     PdfImportedPage page;
     page = writer.getImportedPage(reader, page_number);
     writer.addPage(page);
     document.close();
     writer.close();
     return p;
}
```

```
public static PDFItem concatenate(String file1, String file2,
String outfile) {
      try {
          int pageOffset = 0;
          ArrayList master = new ArrayList();

          Document document = null;
          PdfCopy  writer = null;

          String tmpfile =
CodeGen.randomTmpFileNameGen(CodeGen.TMPPREFIX);

          // we create a reader for the first file

          PdfReader reader = new PdfReader(file1);
          reader.consolidateNamedDestinations();



          // step 1: creation of a document-object
          document = new
Document(reader.getPageSizeWithRotation(1));
          // step 2: we create a writer that listens to the
document
          writer = new PdfCopy(document, new
FileOutputStream(tmpfile));
          // step 3: we open the document
          document.open();


          // we retrieve the total number of pages
          int n = reader.getNumberOfPages();
          List bookmarks = SimpleBookmark.getBookmark(reader);
          if (bookmarks != null) {
              if (pageOffset != 0)
```

```
                SimpleBookmark.shiftPageNumbers(bookmarks,
pageOffset, null);
            master.addAll(bookmarks);
        }
        pageOffset += n;

        // step 4: we add content
        PdfImportedPage page;
        for (int i = 0; i < n; ) {
            ++i;
            page = writer.getImportedPage(reader, i);
            writer.addPage(page);
        }

        PRAcroForm form = reader.getAcroForm();
        if (form != null)
            writer.copyAcroForm(reader);

        // we create a reader for the second file
        reader = new PdfReader(file2);
        reader.consolidateNamedDestinations();
        // we retrieve the total number of pages
        n = reader.getNumberOfPages();
        bookmarks = SimpleBookmark.getBookmark(reader);
        if (bookmarks != null) {
            if (pageOffset != 0)
                SimpleBookmark.shiftPageNumbers(bookmarks,
pageOffset, null);
            master.addAll(bookmarks);
        }
        pageOffset += n;

        // step 5: we add content
        for (int i = 0; i < n; ) {
            ++i;
            page = writer.getImportedPage(reader, i);
```

```
            writer.addPage(page);
        }
        form = reader.getAcroForm();
        if (form != null)
            writer.copyAcroForm(reader);

        if (!master.isEmpty())
         writer.setOutlines(master);
        // step 6: we close the document
        document.close();

        File tmpfile_f = new File(tmpfile);
        File outfile_f = new File(outfile);
        tmpfile_f.renameTo(outfile_f);

        return new PDFItem(outfile);
    }
    catch(Exception e) {
         e.printStackTrace();
         return null;
    }
 }

public static PDFItem concatenate(String file1, String file2) {
    try {
        int pageOffset = 0;
        ArrayList master = new ArrayList();

        Document document = null;
        PdfCopy  writer = null;

        String tmpfile =
CodeGen.randomTmpFileNameGen(CodeGen.TMPPREFIX);

        // we create a reader for the first file
```

```
            PdfReader reader = new PdfReader(file1);
            reader.consolidateNamedDestinations();



            // step 1: creation of a document-object
            document = new
Document(reader.getPageSizeWithRotation(1));
            // step 2: we create a writer that listens to the
document
            writer = new PdfCopy(document, new
FileOutputStream(tmpfile));
            // step 3: we open the document
            document.open();


            // we retrieve the total number of pages
            int n = reader.getNumberOfPages();
            List bookmarks = SimpleBookmark.getBookmark(reader);
            if (bookmarks != null) {
                if (pageOffset != 0)
                    SimpleBookmark.shiftPageNumbers(bookmarks,
pageOffset, null);
                master.addAll(bookmarks);
            }
            pageOffset += n;

            // step 4: we add content
            PdfImportedPage page;
            for (int i = 0; i < n; ) {
                ++i;
                page = writer.getImportedPage(reader, i);
                writer.addPage(page);
            }

            PRAcroForm form = reader.getAcroForm();
```

```
            if (form != null)
                writer.copyAcroForm(reader);

            // we create a reader for the second file
            reader = new PdfReader(file2);
            reader.consolidateNamedDestinations();
            // we retrieve the total number of pages
            n = reader.getNumberOfPages();
            bookmarks = SimpleBookmark.getBookmark(reader);
            if (bookmarks != null) {
                if (pageOffset != 0)
                    SimpleBookmark.shiftPageNumbers(bookmarks,
pageOffset, null);
                master.addAll(bookmarks);
            }
            pageOffset += n;

            // step 5: we add content
            for (int i = 0; i < n; ) {
                ++i;
                page = writer.getImportedPage(reader, i);
                writer.addPage(page);
            }
            form = reader.getAcroForm();
            if (form != null)
                writer.copyAcroForm(reader);

            if (!master.isEmpty())
             writer.setOutlines(master);
            // step 6: we close the document
            document.close();


            return new PDFItem(tmpfile);
        }
        catch(Exception e) {
```

```
            e.printStackTrace();
            return null;
      }
  }

public static ArrayList<PDFWord> listWordsAndQuads(String
file_path) throws IOException {

      InputStream ip;
      PDFDocument pdfdoc;
      ArrayList<PDFWord> word_list = new ArrayList<PDFWord>();

      ip = new FileInputStream(file_path);
      pdfdoc = PDFFactory.openDocument(ip);

      //pdfdoc.exportAnnotations();
      ListIterator li = pdfdoc.getWords();

      while(li.hasNext()){
          PDFWord word = (PDFWord)li.next();
          word_list.add(word);
      }

      return word_list;
}

public static int[] stringInPdf(String word, PDFItem p) throws
IOException {
     InputStream ip;
     PDFDocument pdfdoc;

     String file_path = p.getPath();
     ip = new FileInputStream(file_path);
     pdfdoc = PDFFactory.openDocument(ip);
     ArrayList<Integer> list = new ArrayList<Integer>();
     //pdfdoc.exportAnnotations();
```

```
    ListIterator li = pdfdoc.getWords();


    while(li.hasNext()) {

    PDFWord p_word = (PDFWord)li.next();
    if (p_word.getString().equals(word))
           list.add(new Integer(p_word.getPageNumber()));
    }
    int array[] = new int[list.size()];
    for(int i = 0; i<list.size(); i++){
    array[i] = list.get(i).intValue();
    }
    return array;
}

public static void highlightWord( String file_path, String word)
throws Exception, DocumentException{
    try{
           InputStream ip;

    PDFDocument pdfdoc;

    ip = new FileInputStream(file_path);
    pdfdoc = PDFFactory.openDocument(ip);

    //pdfdoc.exportAnnotations();
    ListIterator li = pdfdoc.getWords();

    String temp_file_name =
CodeGen.randomTmpFileNameGen(CodeGen.TMPPREFIX);
    PdfReader reader = new PdfReader(file_path);
    PdfStamper stamp = new PdfStamper(reader,
      new FileOutputStream(temp_file_name));
    PdfContentByte under;
    // change the content beneath page 1
```

```
 /* for every word if it matches the search key, get its quad
coordinates and draw
  * a rectangle with them.
  */

     while(li.hasNext()){
         PDFWord p_word = (PDFWord)li.next();

         if(p_word.getString().equals(word)) {

                  under =
stamp.getUnderContent(p_word.getPageNumber());
                  //Calculate the coordinates of the left corner of
the word
                        ListIterator list =
p_word.getQuadList().listIterator();

                        Quad q1 = (Quad) list.next();
                        Quad q2 = q1;
                        //If the word is made of more than one quad
we need the leftmost and rightmost ones coordinates
                        //else use only the first one's

                        while(list.hasNext())
                               q2 =(Quad) list.next();

                        //change the content on top of page 1

                        //TODO: Set color and width

                        //defines the rectangle to be drawn
                        under.rectangle((float) q1.p1().x(),(float)
q1.p1().y(), (float)(q2.p2().x() - q1.p1().x()),(float)
(q1.p4().y() - q1.p1().y()));
```

```
                        //actually draws the rectangle
                        under.stroke();
            } // end if
        } //end while
        stamp.close();
       File f = new File(file_path);
        File tmp = new File(temp_file_name);
        if(!f.delete()){
           System.out.println("Error file"+f.getAbsolutePath());
        }
        tmp.renameTo(f);
}
catch(com.adobe.internal.pdf.cos.util.exceptions.PDFInstantiateExce
ption e){
    System.out.println("Error with pdf format. Only standard pdf
files supported");
}
}

public static void setAuthor(PDFItem p, String author) throws
DocumentException, IOException{
    PdfReader reader = new PdfReader(p.getPath());
    System.out.println("Opening "+p.getPath());
    File tmp = new
File(CodeGen.randomTmpFileNameGen(CodeGen.TMPPREFIX));
    PdfStamper stamper = new PdfStamper(reader, new
FileOutputStream(
                tmp));

    HashMap info = reader.getInfo();
    info.put("Author", author);
    info.put("Creator", "SPML ");
    stamper.setMoreInfo(info);
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    XmpWriter xmp = new XmpWriter(baos, info);
    xmp.close();
```

```
        stamper.close();
        tmp.renameTo(p.getPdfFile());
    }

    public static int pageNum(String file) throws IOException{
        PdfReader reader = new PdfReader(file);
         reader.consolidateNamedDestinations();
         return reader.getNumberOfPages();
    }

    public static void deleteTmpFiles(){
         File f = new File(System.getProperty("user.dir"));
         TmpFileNameFilter filter = new TmpFileNameFilter();
         String[] list = f.list(filter);
         for(int i = 0; i<list.length; i++){
         File foo = new File(list[i]);
         if(foo.exists())
                foo.delete();
         }
    }

}

/**
 * MyPDFFileFilter imlplments the FilanameFilter used for filtering
   in listPDFFiles
 * @author stefy
 *
 */
class MyPDFFileFilter implements FilenameFilter {
    public boolean accept(File f, String name){
        if(name.endsWith(".pdf"))
              return true;
        else return false;
    }
```

```
}

class TmpFileNameFilter implements FilenameFilter {
   public boolean accept(File f, String name){
        if(name.endsWith(".pdf") && name.startsWith(CodeGen.TMPPREFIX))
              return true;
        else return false;
   }
}
```

Run SPML source code

Compile.sh

Running SPML code from the source code to result on Unix/Linux platform

```
#!/bin/bash
#LOGFILE is the file where the test output is written
LOGFILE="SPMLWalkerTesting.log"

# TESTDIR is the path to the directory holding the test files
FILENAME=$1

# ANTLR_PATH is the path of antlr jar library
ANTLR_PATH="/Applications/eclipse/plugins/org.antlr_2.7.6/antlr.jar"

#PROJECTPATH holds the classpath for the project
PROJECTPATH="/Users/stefy/Documents/workspace/SPML-v2/bin"

#EXECPATH is the class of SPML compiler
EXECPATH="spml.Main"

COMPILECMD="java -cp $ANTLR_PATH:$PROJECTPATH:$CLASSPATH $EXECPATH"
COMPILECMD2="javac -cp $ANTLR_PATH:$PROJECTPATH:$CLASSPATH"

$COMPILECMD $FILENAME
```

```
$COMPILECMD2 $(basename "$FILENAME" .spml).java
$COMPILECMD $(basename "$FILENAME" .spml).java
```

Main.java

Running SPML code from the source code to result

```
package spml;

import java.io.*;

import antlr.CommonAST;
import antlr.debug.misc.ASTFrame;
/**
 *
 * @author Dhivya and Jayesh
 */
 class Main {

    /** Creates a new instance of Main */
    public static void main(String[] args) {
        // TODO code application logic here

       SpmlOut outxx = null;
        try {
             //String str;
//            SPMLLexer lexer = new SPMLLexer(new
  DataInputStream(System.in));
       String file = args[0];
       System.out.println(file);
        if(!file.endsWith(".spml")){
             System.err.println("Invalid file type!");
             System.exit(1);
        }
        SPMLLexer lexer = new SPMLLexer(new DataInputStream(new
  FileInputStream(args[0])));
       SPMLParser parser = new SPMLParser(lexer);
```

```
        parser.setASTNodeClass("spml.SpmlAST");
        parser.program();

            // Get the AST from the parser
            CommonAST parseTree = (CommonAST)parser.getAST();

            // Print the AST in a human-readable format
 //           System.out.println(parseTree.toStringList());
            //System.out.println(parseTree.toStringTree());

            // Open a window in which the AST is displayed
   graphically
//            ASTFrame frame = new ASTFrame("AST from the Simp
   parser", parseTree);
 //           frame.setVisible(true);

            SpmlOut.initFileName(args[0]);
            SPMLTreeWalker walker = new SPMLTreeWalker();
            walker.setASTNodeClass("spml.SpmlAST");
            walker.program(parseTree);

            //Call codegen.g
            System.out.println("Starting code generation...");
            SPMLCodeGen cg = new SPMLCodeGen();
            cg.program(parseTree);
            cg.out.writeToJava();
//            outxx = cg.out;
//            System.out.println(outxx.getOut().toString());
           System.out.println("Executing the java outputs: ");
        } catch(Exception e) {
            System.err.println(e);
            System.exit(0);
        }
   }
}
```

SPML.bat

Batch program to run SPML code from the source code to result on Windows platform

```
@echo off
set CLASSPATH=%CLASSPATH%;E:\MyDownloads\JAVA Latest
   versions\eclipse\plugins\org.antlr_2.7.6\antlr.jar;E:\Columbia
   Spring 2007\PLT Project\SPML\xpaaj_sdk\XPAAJ.jar;E:\Columbia Spring
   2007\PLT Project\SPML\itext-2.0.0.jar;E:\final_plt
cd spml
javac *.java
cd ..
java spml.Main %1
javac %2
java %3
cd spml
del *.class
cd ..
del *.class
del *.java
```

Test

test-walker.sh

Shell script for lexer / parser / tree walker of SPML

```
#!/bin/bash
#LOGFILE is the file where the test output is written
LOGFILE="SPMLWalkerTesting.log"

# TESTDIR is the path to the directory holding the test files
TESTDIR=$1

# ANTLR_PATH is the path of antlr jar library
ANTLR_PATH="/Applications/eclipse/plugins/org.antlr_2.7.6/antlr.jar"
```

```
#PROJECTPATH holds the classpath for the project
PROJECTPATH="/Users/stefy/Documents/workspace/SPML-v2/bin"

#EXECPATH is the class of SPML compiler
EXECPATH="spml.Main"

rm $LOGFILE
touch $LOGFILE

COMPILECMD="java -cp $ANTLR_PATH:$PROJECTPATH:$CLASSPATH $EXECPATH"
echo $COMPILECMD
for i in `ls $TESTDIR*.spml`
do
   echo"********** $i*********" >> $LOGFILE;
   $COMPILECMD $i 2>&1 | grep line | grep -v Exception >> $LOGFILE;
   $COMPILECMD $i 2>&1 | grep Exception >> $LOGFILE
done;

less $LOGFILE
```

Testing Lexer / Parser / Walker

Our team generated many test files for the part. These are some examples of test files in SPML.

Testing programs

```
   program1.spml

   start(){
      string path;
      path ="/Users/stefy/Desktop";
      pdf p[0];
      pdf result;
      result = create "result.pdf";
```

```
   p = pdf in path;

   int l, index;
   result = extractpage p[0] 1;
   index = 1;
   l = length p;

   while( index < l ){
    pdf new;
    new = extractpage p[index] 1;
    result = result + new;
    index = index+1;
   }
}
```

program2.spml

```
pdf create_catalogue(pdf p[]);

start(){
   string path;
   path ="/Users/stefy/Desktop";
   pdf p[0];
   pdf result;

   p = pdf in path;

   result = create_catalogue(p);
}

pdf create_catalogue(pdf p[]){
          int l, index;
        pdf result;
   result = extractpage p[0] 1;
        index = 1;
```

```
        l = length p;

        while( index < l ){
                pdf new;
                new = extractpage p[index] 1;
                result = result + new;
                index = index+1;
        }
  return result;
}
```

Testing types

```
type-good1.spml

start(){

string s;

s = "ciao";
int a;
a= 1;
string t;
t = s+a;
}

type-good2.spml

start(){

string s;

s = "ciao";
int a;
a= 1;
string t;
```

```
t = a+s;

}
```

Testing if

```
conditiona-good2.spml

int foo();

start(){
 if(foo() == foo()){}
}

int foo(){
 return 1;
}
```

```
conditiona-good3.spml

string foo();

start(){
 if(foo() == foo()){}
}

string foo(){
 return "ciao";
}
```

Testing while

```
while-good1.spml

start(){
while(1){
```

```
print 1;
}
}
```

while-good2.spml

```
start(){

int a;
a = 1;

while(1){
 if(a != 0) {
  break;
 }
 print 1;
}

}
```

Testing print

print-good1.spml

```
start()
{
int a;
a = 7;
pdf p;
 p = "ejeje";
print "hehee";
print a;
print (a+5)*6;
}
```

print-good2.spml

```
start()
{
int a;
a = 7;
pdf p;
 p = "ejeje";
print "hehee";
print a;
print "hello" + "jwjwjw";
string s;
s = "jeje";
print s+s+s+8;
}
```

Testing in

```
in-good1.spml

start(){
int a;
a = "ciao" in "baubau";
}

in-good2.spml

start(){
int a;
string s,t;
s = "ciao";
t = "baubau";
a = s in t;
}
```

Testing functions

function-good1.spml

```
/* recursion */

int foo(int a);

start() {

int a;
a = foo(5);
print a;

}

int foo(int a){
 if (a == 0){
  return 0;
 }
 else {return a+foo(a-1);}
}
```

function-good2.spml

```
int foo(int a, int b);

start(){

int a,b,c,d,e;

a = 0;
b = 1;
d = 0;
e = 12;

c = foo(foo(a,b), foo(d,e));
```

```
print c;
}

int foo(int a, int b){
 return a+b;
}
```

Tesing Code Generation

Our team generated many test files for the part. These are some examples of test files in SPML.

cg-test26.spml

```
string[] foo(int a, string b);

start(){
foo(3,"Asd");
}

string[] foo(int asd, string basd){
        string s[4];
        s[0] = basd;
        s[1] = basd;
        s[2] = basd;
        s[3] = basd;
        return s;
}
```

cg-test27.spml

```
int[] foo();

start(){
int a[0];
foo();
```

```
}

int[] foo(){
int b[10];
int index;
index = 10;
while(index > 0){
  b[index] = index;
}
return b;
}
```

cg-test28.spml

```
int[] foo();

start(){
int a[0];
a = foo();
int b;
b = length a;
while(b >0){
b = b-1;
print a[b];
}
}

int[] foo(){
int b[10];
int index;
index = 9;
while(index > 0){
  b[index] = index;
 index = index-1;
}
return b;
```

```
}
```

cg-test29.spml

```
string getWord();
start()
{
pdf final;
string path;
path = "F:\\PLT-prj files\\Testing\\getwordpdfs.pdf";
final = create path;
pdf dirPdfs[10];
dirPdfs = pdf in "F:\\PLT-prj files\\Testing\\pdfs" ;
int lenDir;
lenDir = length dirPdfs;

if(lenDir!=0)
{
int i;
i = 0;
while(i<lenDir)
{
pdf temp;
temp = dirPdfs[i];

int j;
int numOfPages;
j = 0;
numOfPages = length temp;
while(j < numOfPages)
{
pdf temp2;
temp2  = extractpage temp j;
int a[100];
a = getWord() in temp2;
int y;
```

```
y = length a;
if(y)
{
int lenfinal;
lenfinal = length final;
if(lenfinal==0)
{
final = extractpage temp j;
}
else
{
final = final + temp2;
}
```

cg-test30.spml

```
start(){
pdf p1, p2;
p1 = "E:\\plt pdfs\\T1.pdf";
p2 = "E:\\plt pdfs\\T2.pdf";
p1 = p1+p2;
}
```