The WPSM Language:
A simple language that transform simple data structure into complex xML format

Wai Y. Wong wyw2101@columbia.edu
Peter Chen pc2106@columbia.edu
Seema Gupta sg807@columbia.edu
Miqdad Mohammed mm1723@columbia.edu

COMS W4115, Programming Language and Translators, Spring 2003
Instructor: Prof. Stephen Edwards
TA: Michael Locasto
May 13, 2003

Table of contents:

Appendix B.

Appendix C.

Appendix D.

1. Introducing WPSM

1.1. Introduction

The WPSM language and environment is designed to help a content manager to transform a mass amount of records from a simple data file into a complex XML file format. (See Appendix A) Through a few simple lines of WPSM codes, the user will be able to manipulate the input data and generate a well-formed and valid XML file containing that data. More importantly, the simple WPSM syntax will also manage the complex XML syntax for the user. Using WPSM language, the content manager will not have to worry about the complex XML syntax, such as declaration and tags, at all.

The WPSM language supports Extensible Markup Language (XML) 1.0 (Second Edition) and is designed to be easy to learn, productive, robust, portable, and Internet ready. The Java integration and WPSM Virtual Machine allows the WPSM program to be created and executed in all environments that Java supports.

1.2. Background

XML, Extensible Markup Language, is designed as a means of describing, structuring, storing and sending information or data and is derived from SGML, Standard Generalized Markup Language. It is introduced to make up for the main limitation of HTML, which is the inability to create your own markup.

XML provides a standard self-documented format to exchange information among software components and therefore its documents can replace ASCII files and formats like CSV, comma-separated values. Because XML is able to encode information that is nested and in lists, its documents can replace relational SQL-like encoding of data by providing a more "object-like" encoding of data.

Today, in e-Business, XML is wildly used in supporting mass content management. In the data warehouse, it eases the transferring of data between two systems that have different schema structures. In the Business-to-Business (B2B) model, it allows for the exchange of a large amount of data between companies and organizations. For example, a company could develop an application, based on XML specifications, to exchange financial information.

1.3. Motivation

XML is a markup language for documents containing structured information. This markup language is a mechanism to identify structures in a document. The XML specification defines a standard way to add markup to documents. Structured information contains both contents and some indications of what role that the contents play. However, because of the standards and rules of XML, it is relatively difficult for the users (e.g. content managers, who do not

know how to write program) to implement and maintain the XML code correctly and efficiently.

Now, with WPSM, the users do not have to know about the XML rules and syntax at all. They can create a well-formed and valid XML file in three steps. (Here we use a CSV file format as an example) First, the user creates a CSV file using a spreadsheet program, e.g. MS Excel. The users simply enter all desired data in a spreadsheet, and they save the spreadsheet as a CSV file. Secondly, the users write a few line of WPSM code and compile the code. Now, the users' job is almost done. The last step is to run the program the WPSM just compiled. By running this WPSM compiled program, a well-formed and valid XML file would be created automatically given the inputted CSV file. This XML file is ready to be used for any purpose.

Users are not limited to map and translate from one simple data structure file to an XML file. WPSM gives the users the flexibility and usability to manipulate the content of data and strings and customize the generated XML file to suit their purpose. With some simple WPSM syntax and techniques, they would be able to output their desired contents and formats into an XML file.

The simplicity of WPSM will allow a user new to programming to become a skilled WPSM programmer. All that is necessary to master the language is to breeze through the reference manual provided.

1.4. Goals

Ease of use

WPSM is designed for people who may never have done any programming in their lives. The user will be able to begin coding WPSM as soon as they read the WPSM reference. As opposed to most other programming languages, the user does not have to understand complicated concepts, such as object-oriented programming, data type structure, allocation and freeing of memory, pointers, and memory address. All they need to understand is the simple WPSM syntax and keywords.

Productivity

Use simple algorithms to manage mass amounts of data and manipulate complex XML syntax. Writing few lines of WPSM codes, users can organize and convert thousands of records from a data file in some simple format, such as CSV, to a well-formed XML format. For example, with a small WPSM program, the companies will be able to easily transform the whole system records from one data schema into another. Furthermore, they can maintain and manipulate the contents easily by simply modifying a bit of the WPSM code whenever necessary.

Robust

Because WPSM's syntax is so simple and is capable of generating a complex output, reliability is of the utmost important. WPSM allows extensive compile-time checking so bugs can be caught early. Ideally, as long as the users compile their code successfully, users will not have to deal with run time errors. This is why WPSM gives the users a robust and reliable environment. It is one of our goals to make non-programmers become well-skilled WPSM programmers.

Portable

A program written in WPSM can run in all environments, and is not limited to any single environment. Since WPSM is implemented using Java, it is truly portable. It is an architecture neutral language that supports a variety of systems, CPUs, and operating systems. Since WPSM is compiled in JAVA and generates JAVA, it is as portable.

Internet ability

Because WPSM is an architecture neutral language, the compiled code can run in any web-based environment. This helps users working in different locations around the world to be able to share and access the compiled code. For example, a content manager in a New York warehouse can access the stocking database in a California warehouse through the Internet.

1.5. Features

Filtering

WPSM allows a user to filter through a set of data using the functions provided. A user could potentially open a file, and run through it line by line, specifying which column is to be searched for what value. If the data includes the specified value, the user will be able to state what the resulting action should be (i.e. include the resulting data in the generated XML file).

Column manipulation

As an added benefit, the users can include/exclude columns in/from the generated XML file. The user can create a new column to add to the data, and decide upon a default value for every row in that column. Alternatively, the user has the option of excluding a column from the resulting XML file, by specifying the position in the input file to exclude.

XML tags (element, attribute) manipulation

One of the major features of this language is that it allows a user to indicate the name of the different elements and tags. If, for example, a user is compiling a list of books and all the attributes associated with those books, they can set the name of the tags at each position. (Please see **Sample Syntax** for example).

Data manipulation

WPSM allows a user to concatenate values of two different variables. For example, they may want to concatenate the values of var1 and var2 or var1 and column1. These actions would result in var1var2 and var1column1. (Please see **Sample Syntax** for example).

Arithmetic

WPSM provides the user with all the basic arithmetic operations. They will be able to add, subtract, and multiply any number of variables and columns.

Error handling ability

If the compiled program generates an error when it is run, the user can specify, within the code, the action to take once that error is seen. This will allow the user to account for any foreseen errors, and allow the program to continue to generate code despite the occurrence of invalid data.

1.6. Sample Syntax

Basic Syntax

```
START                    // Start of program

END                      // End of program

$[Integer]               // Position of the record. $1, $2,..,$n

$[Letters]               // WPSM data-type variable

IF (bool_exprs) {        // If statement
      stmt_list
}
ELSEIF (bool_exprs) {
      stmt_list
}
ELSE {
      stmt_list
}

WHILE (bool_exprs) {     // While statement
      stmt_list
}
```

XML tags (element, attribute) manipulation

```
//WPSM code
$RN := "Book"            // Sets the category (XML root element)
                         // name
$1 := "Title";
$2 := "Author";
$3 := "Publisher";
$4 := "Price";
$5 := "Date";
```

```
//sample CSV data:
Computer Graphics, James D. Foley, Addison-Wesley, $69.95, 1996

//Would be converted to:
<Book Title = Computer Graphics Author = James D. Foley Publisher =
Addison-Wesley Price = $69.95 Year = 1996 />
```

Filtering

```
//WPSM code
START;
$string := "Java";       //assign value to a variable
IF ($1 == $string)
        ; //then "skip stmt" don't create the record in XML
END;

//sample CSV data:
Computer Graphics, James D. Foley, Addison-Wesley, $69.95, 1996
Thinking in Java, Bruce Eckel, Prentice Hall PTR, $49.99, 2002

//Only one record will be converted to:
<Book Title = Computer Graphics Author = James D. Foley Publisher =
Addison-Wesley Price = $69.95 Year = 1996 />
```

Data manipulation

```
$var1 = "gold"
$var2 = "rush"
$var3 = $var1 + $var2

//the resulting value of $var3 would be "goldrush"

$var1 = "gold"
$var3 = $var1 + $1

//the resulting value of $var3 would be "gold" concatenated with the value of
//position 1 of that line
```

1.7. Summary

WPSM is a powerful scripting language that allows the user to transform mass amount of data records from one simple structure to the complex XML format. It also provides the user with the ability to manipulate columns of data, strings, and numbers in the creation of a well-formed and valid XML output. It handles all XML rules, such as DTD, root element, element, and attribute. It eliminates the Content Manager's trouble in dealing with XML syntax while managing large data files, and allows them to focus on their job (Content Management).

2. Reference Manual

2.1. Syntax notation

Syntax notations are defined as they are introduced in this document. Nonterminal symbols are indicated by *italic* type, and terminal symbol are indicated by gothic type. The symbol followed by ‘ * ’ denotes the symbol may be occur zero or more times. The symbol followed by ‘ + ‘ denotes the symbol may occur one or more times. The symbol followed by ‘ ? ‘ denotes the symbol is optional, which may occur zero or one time. ‘ (s1 | s2) ‘ denotes a choice between the symbol sequence s1 and the symbol s2.

2.2. Lexical conventions

There are several kinds of tokens used in WPSM: identifiers, keywords, constants, strings, separators, and variables. In general blanks, tabs, new lines, and comments are ignored except in the case where they serve to separate tokens. Different adjacent identifiers, constants, and certain operator-pairs must be separated by these characters.

2.2.1. Comments

One of the Java style comments is supported. The characters // introduce a comment, which terminates with the new line. // has no special meaning inside comment line.

2.2.2. Identifiers (names)

An identifier is a sequence of letters and digits; the first character must be alphabetic. Upper and lower case letters are considered different.

Identifier → letter *(* letter | digit | *‘_’)**

2.2.3. Keywords

The following identifiers are reserved for use as keywords, and may not be used otherwise:

BEGIN
WHILE
READLINE
IF
ELSEIF
ELSE
PRINT
CREATRECORD
AND
OR
EOF
END

2.2.4. Constants

There are several kinds of constants, as follows:

2.2.4.1. *integer* constants

An *integer* constant is a sequence of *digits*.

integer → *digit*+

2.2.4.2. *string* constants

A string is a sequence of ACSII characters surrounded by double quote ". A string has the Java type string (see Java reference).

string → " *character*+ (*character* | *digit*)* "

2.2.5. Separators

The following ASCII *characters* are separators:
{
}
;
,
(
)
[
]

2.2.6. Variables

There are three types of variables introduced in this document. Regular variables and attribute position variables must begin with ' $ ' symbol, otherwise an variable declaration error would be complained. The regular variables must be initialized before they are being used.

2.2.6.1. Regular variables

The variable name must begin with ' $ ' symbol and followed by at least one alphabetic, then any combination of alphabetic, integer number and /or underscore '_' is optional.

letter: 'a'..'z' | 'A'..'Z';
int: ('0'..'9')+;
var: '$' letter (letter | int | '_')*;

2.2.6.2. Attribute position variable

The attribute position variables must begin with ‘ $ ‘ and followed by at least one or more *digits* (*integer*). This variable is bounded to the position of the record attribute and has direct reference to the attribute value.

attr_var: ‘$’ (‘0’..’9’)+;

2.2.6.3. Built-in variables

The following identifiers are reserved for use as built-in variables, and may not be used otherwise:

FN	input file’s filename
OFN	output file’s filename
FS	field separator
RT	XML root element
RN	record name (XML element name)
RA	record attribute (XML attribute name)
RV	record value (XML value)
RV[*integer*]	particular attribute value

Integer type:

RA_SIZE	returns the number of record attributes
RV_SIZE	returns the number of record values

2.3. Expression

The precedence of expression operators is the same as the order of the major subsections of this section (highest precedence first). Within each subsection, the operators have the same precedence. Left- or right-associativity is specified in each subsection for the operators discussed therein.

2.3.1. Primary expressions

2.3.1.1. *identifier*

An *identifier* is a primary expression. Its type is specified by its declaration.

2.3.1.2. *integer* constant

An *integer* constant is a primary expression.

2.3.1.3. *string* constant

A *string* is a primary expression.

2.3.1.4. (*expression*)

A parenthesized *expression* is a primary expression whose type and value are identical to those of the unadorned expression.

2.4 Operators

2.4.1 Additive operators

The additive operators + and – group left-to-right.

2.4.1.1 *expression* + *expression*

The result is the sum of the expressions.

2.4.1.2 *expression* – *expression*

The result is the difference of the operands.

2.4.1.3 *expression* * *expression*

The result is the multiplication of the operands.

2.4.2 Logical operators

2.4.2.1 *expression* OR *expression*

2.4.2.2 *expression* AND *expression*

The use of "AND" and "OR" logical operators are identical to "&&" and "||" operators respectively in C/C++ and Java.

Logical_And_Expr: Logical_expr AND Logical_expr
Logical_And_Expr is true if both logical expressions are true.

Logical_Or_Expr: Logical_expr OR Logical_expr
Logical_Or_Expr is true if either one or both of logical expression(s) is/are true.

2.4.3 Equality operators

2.4.3.1 *expression* == *expression*

2.4.3.2 *expression* != *expression*

The = = (equal to) and the != (not equal to) operators are exactly analogous to the relational operators except for their lower precedence.

2.4.4 Assignment operators

2.4.4.1 *variable* := *expression*

The := (assign to) operator groups right-to-left. The value of the expression replaces the object referred to by the variable. The operands need not have the same type, but must be either integer or string constants.

2.4.4.2 *variable* += *expression*

The behavior of this additive assignment is equivalent to " $var1= $var1 + *expression* " while $var1 is evaluated only once.

2.4.4.3 *variable* -= *expression*

This is similar to section 4.2.3.2. The behavior of this additive assignment is equivalent to " $var1 = $var1 – *expression* " while $var1 is evaluated only once.

2.5 Declarations

There is no declaration needed. The variable is automatically assigned a data type according to the type of the value. For example:

$var1 := "hello";

The variable " $var1 " is assigned with the value "hello" and the data type for this variable is automatically declared as a string.

$var2 := 12;

The variable " $var2 " is assigned with the value 12 and the data type for this variable is automatically declared as an integer.

2.6 Statements

Except as indicated, statements are executed in sequence.

2.6.1 Expression statement

Most statements are expression statements, which have the form *expression*;

2.6.2 Compound statement

Several statements can be used where one is expected, the compound statement is provided:

compound-statement:
{ *statement-list* }

statement-list:
statement
statement statement-list

2.6.3 Conditional statement

The four forms of the conditional statement are

if (*expression*) *statement*
if (*expression*) *statement* else *statement*
if (*expression*) *statement* elseif (*expression*) *statement*
if (*expression*) *statement* elseif (*expression*) *statement* else *statement*

In all cases, the expression is evaluated, and if it is non-zero, the first sub-statement is executed. In the third and fourth cases, if the elseif expression is non-zero, then the sub-statement in the elseif is executed. In the second and fourth cases, if both the if and elseif expressions are zero, then the sub-statement in the else is executed. As the " if " condition in Java, the "else" ambiguity is resolved by connecting an else with the last encountered elseless if or elseif. Zero or more elseif case can be used in a conditional statement.

2.6.4 Loop statement (while)

The while statement has the following form:
while (*expression*) *statement*

The sub-statement is executed repeatedly as long as the value of the expression remains non-zero. The test takes place before every execution of the statement.

2.7 Scope rules

All components of the WPSM program must be part of one file and must be compiled at the same time. Not all source code in a program will be compiled and generated into target language. All variables are declared as global variables. Therefore, there are two kinds of scopes to consider: first, the lexical scope of the variable; and second, the scope of the dependency.

2.7.1 Lexical scope

WPSM can be considered similar to a scripting language, and also uses block structuring. Variables declared at any level are globally visible, as WSPSM uses global scoping. Variable types can be changed dynamically based on the type of the value assigned to it.

$var1 := "hello";
$var1 := 12;

The first case shows that " $var1 " is declared as string type and assigned a value "hello". The second case re-declares the same variable to an integer with the value 12.

2.7.2 Scope of dependency

WPSM built-in variables and reserved keywords (built-in functions) have some dependencies on one another. WPSM built-in variables are designed to support this scope. Minimal initiations are needed to prepare the parameters before executing the built-in functions. If a built-in variable has a default value, it becomes optional to initialize this variable. The WPSM Java library contains built-in functions that are repeatedly used by a program. The precedence of some built-in variable (see built-in variables for variable specification) and built-in functions is indicated below:

dependency:
    primary

primary:
    built-in variable
    FN
    READLINE( )
    OFN

built-in variable
    FN (has default "*")
    FS
    RT (has default "ROOT")
    RN
    RA
    RV
    RV[*integer*]
    RA_SIZE
    RV_SIZE

3 Install WPSM

3.6 System requirement

WPSM programming language (WPSM) is implemented and tested on Java 2 Platform, Standard Edition (J2SE) version 1.4.1_02; therefore WPSM will work perfectly on any environment that contains this Java SDK.

Note: However, our research and development team designs, implements, and tests the latest WPSM code under the Window NT/2000/XP OS environment; therefore, we are only supporting our code under these OS environments. If the user wants to use WPSM in the platform other than Windows 2000 NT/2000/XP, such as UNIX, we are not reliable for the unexpected errors.

To use the WPSM programming language, you need to have the following development environment:

Windows NT/2000/XP
Java 2 Platform, Standard Edition (J2SE) version 1.4.1_02
Note: please visit http://java.sun.com/j2se/1.4.1/ for more Java information

3.7 Install WPSM package

Note: please see the readme.txt in the WPSM package for the latest information.

1. Un-zip the WPSM_version.zip on your machine under C drive
2. Un-zip will create a WPSM folder under C drive
3. You should have all the following directories under WPSM folder:
   bin --> Contains all java classes for WPSM
   sample --> WPSM examples
   src --> WPSM source code

3.8 Setup WPSM environment

Note: please see the readme.txt in the WPSM package for the latest information.

1. Setup the system "PATH"
   on Command Prompt windows, set PATH=%PATH%;C:\WPSM\bin
   or
   Add "C:\WPSM\bin" in system PATH in the system environment variable.
2. Setup the system "classpath"
   on Command Prompt windows, set classpath=%classpath%;C:\WPSM\bin
   or
   Add "C:\WPSM\bin" in system CLASSPATH in the system environment variable.

4 Tutorial

We provide a “simple.xf” WPSM source code for the tutorial section. **Please see Appendix A. for the complete source codes and comments.** You can find this sample WPSM file in “C:\wpsm\sample” directory on your machine.

“sample.xf” provides the complete comments for you to understand each WPSM syntax. Carefully reading the comment should be able to help you to know what the whole program is doing.

“sample.xf” WPSM program simply transforms a simple data structure input file, “fam.txt”, into a complex XML output file, “fam.xml”. As you can see in the WPSM syntax in “sample.xf”, only few lines of WPSM codes able the inventory manager to generate a well-formed XML.

Here is how to run “sample.xf” in your WPSM environment:
1) Open a Comment Prompt window
2) Got to "C:\wpsm\sample" directory
3) Compile the WPSM source code and run it at same time.

```
C:\wpsm\sample\>wpsm sample
```

4) Done

Note: very simple, isn’t it? We made to run WPSM program is just as simple as run a perl program. This is our goal since we begin to design our program, make it simple and ease of use. There is another way to run the sample WPSM program.

3) Compile the WPSM source code, “sample.xf”. WPSM compiler will generate a target java source file called “sample.java”

```
C:\wpsm\sample\>wpsmc sample
```

4) Compile the target java code with javac. Java compiler will generate a java class file called “sample.class”.

```
C:\wpsm\sample\>javac sample.java
```

5) Run compiled WPSM program, “sample.class”, to transform “fam.txt”, a simple data structure file, to “fam.xml”, a complex XML file.

```
C:\wpsm\sample\>java sample
```

6) You should have “fam.xml” in the sample directory now. Compare the different between “fam.txt” and “fam.xml”.
7) Done

5 Project Plan

5.6 Team responsibility

The whole WPSM project is break into 4 major roles during the research and development phases.

Project management & system architecture design
Peter T. Chen is responsible for: WPSM architecture design, define the project objective, define the initial WPSM syntax, define the test plan, development environment decision, system infrastructure and integration, team and project management, regular meetings setup, team communication, assign the job, and preparation and submit the reports.

Front-end design & implementation
Wai Y. Wong is responsible for: front-end subsystem detail design, WPSM syntax and semantics, WPSM parser and lexer implementation on ANTLR, generate Abstract Syntax Trees (AST), help the back-end developer to handle WPSM AST, support the tester for WPSM syntax testing, and providing the front-end documentation to the final report.

Back-end design & implementation
Miqdad Mohammed is responsible for: back-end subsystem detail design, WPSM AST tree walker, generating the target language (JAVA) source code, make request to the WPSM library developer, support the tester for WPSM features testing, and providing the back-end documentation to the final report.

WPSM library design & implementation and testing
Seema Gupta is responsible for: WPSM library detail design and implementation, supporting the back-end developer on dealing with repeating functions, and features, WPSM testing, WPSM tutorials, and providing the WPSM library and testing documentation to the final report.

5.7 Development phase

During the whole period of WPSM research and development cycle, several design and development phases are introduced. WPSM project is designed and implemented with Software Engineer approach. Couple of implementation cycles and releases is happened during these phases.

Phase I
Introduce to the team members
Team organization, regulation, and regular meeting
Project objective brainstorm

Phase II
Make decision on the project objective
Break down and assign the roles
Education section on the technologies that are needed for the project goal
System requirements gathering

Phase III
White paper
High-level architecture design
Subsystem detail design and discussion
Functional specifications determination

Phase IV
Development environment decision
Subsystem implementation begin

Phase V
Reference manual
Early WPSM system with basic function
WPSM testing begin
First WPSM release

Phase VI
System requirement gathering for WPSM advance features
Subsystem implementation for the advance features
Testing for the advance features
Latest WPSM release

5.8 Software development environment (SDE)

Development platform: Windows 2000/XP
Parser, lexer, and tree walker: ANTLR v2.7.2
WPSM Compiler: implemented in Java SDK v1.4.1
Target language: Java source code

5.9 Project log

February 6th, first group meeting for introduction and brain storming of ideas
February 8th, sent ideas to professor
February 13th, first meeting with TA explaining our idea for the language; also meeting for writing the white paper.
February 15th, WPSM design of architecture
February 18th, meeting cancelled due to weather; white paper submitted
February 22nd, determine WPSM syntax and create some pseudo-code
February 25th, regular meeting with TA and group meeting
February 27th, determine front-end architecture
March 5th, WPSM syntax finalized
March 26th, ANTLR source code completed; LRM written
March 27th, LRM submitted
April 1st, advanced features for front-end started
April 8th, ANTLR source code updated; discussed back-end implementation; questions about library; advanced features
April 15th, library draft and first walk of tree-walker and symbol table
April 26th, put all pieces together... testing level 1
April 27th, found first two errors

April 28th, updated system; questions about non-determinism
May 1st, final report; testing plan; robust final release
May 3rd, dependency check
May 4th, final report draft; first release; java-doc for library made available
May 6th, updated version with advanced features
May 8th, second release; syntax testing completed
May 11th, testing complete; final project and presentation

5.5 Front-End Development Log

March 8th Start writing antlr's source code after reading the antlr's document, create header.
Lexer: Create a few basic lexer grammars.
Parser: Create the startRule grammar.

March 10th Add compile instruction to header.
Lexer: Add wpsm variable grammar, which start with '$' character.
Parser: Add debugging info for startRule grammar.

March 11th After questioning TA, protected rule is applied.
Parser: Read more document, and learn antlr's expression grammar.
TreeParser: Read more document, and learn how antlr's tree parser works.

March 21st Parser: Add assignment grammars and boolean expression grammar.
TreeParser: Play with some tree parser grammar.

March 22nd Lexer: Add a few lexer grammars.
Parser: Add more assignment grammar and start working on "if elseif else" statement grammar.
TreeParser: Still playing with some tree parser grammar.

March 23rd Parser: Start with "while" statement grammar and build antlr's AST

March 25th Parser: Modified some grammars.

March 27th Parser: Start with wpsm build-in variables and attribute variable grammar.

March 30th Parser: Modified some grammar, and learn how protected rule works in parser.

March 31st Parser: Start with wpsm build-in functions and add grammars for increment and decrement.

April 4th Parser: Start work on own ErrorHandler instead of antlr's defaultErrorHandler.

April 5th Parser: Modified some grammars and add multiple boolean expression grammar, using newly added keywords, "AND" and "OR".

April 11th Parser: Work on more wpsm advanced build-in function statement grammar.

April 21st Parser: Continued modified grammars

April 25th Parser: Add more grammars for different variable types, it's getting much more solid.

April 26th Parser: Changed some grammars according to the change of wpsm syntax. This is pretty much finalized.

6 Architectural Design

6.6 WPSM system, overall

The WPSM source code, ".xf", is sent to WPSM compiler. WPSM compiler will generate a Java source file. Then Java compiler will compile this Java source again and generate a Java class. Finally, Java Virtual Machine (JVM) alone with WPSM Library will execute the Java class. At this time the input simple text format file will be convert to a complex XML file. Diagram A. shows the complete WPSM system workflow.

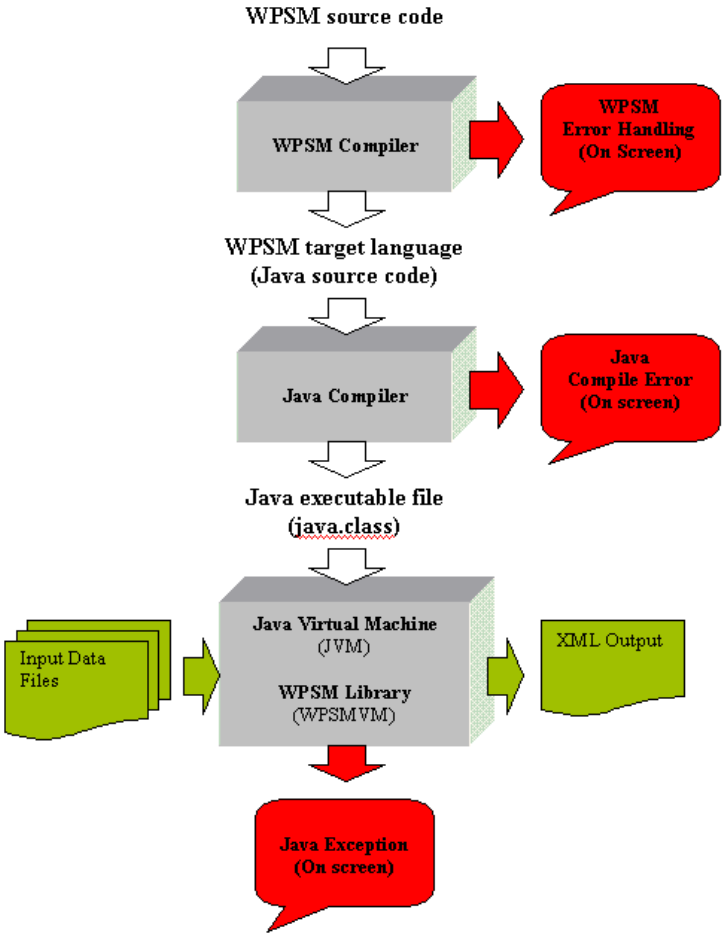

Diagram A. WPSM Architecture

WPSM compiler is broken down into three subsystems, Front-end subsystem, Back-end subsystem, and WPSM library. WPSM library is supporting the first two subsystems on the features they try to accomplish. It supports the WPSM target code at run time. Front-end subsystem contains the WPSM parser and Lexer, and Back-end subsystem contains AST tree-walker and WPSM target language (Java) generator. Diagram B. represents the WPSM compiler system.

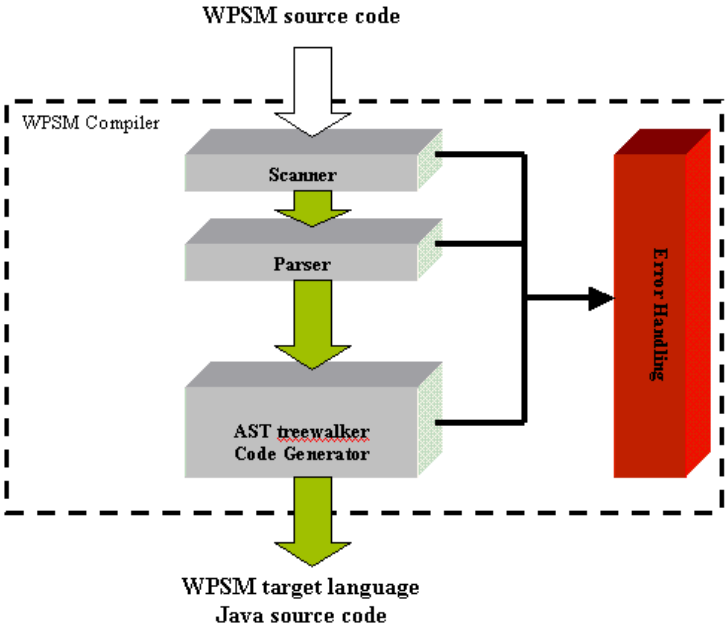

Diagram B. WPSM compiler architecture

6.7 Front-end subsystem

The Front-End Subsystem begins with the creation of a WPSM source file. This file has the extension .xf, and is taken in by the WPSM compiler. The compiler checks the integrity of the code, first by taking in the stream of characters from the source file, and breaking that stream up into tokens using antlr's Lexer Class. Each token consists of a keyword, a variable, an integer, a record attribute, or a record variable (the formats of each of these is explained above). Once these tokens have been created, they are sent to antlr's Parser Class. The Parser class takes these tokens and checks whether they are syntactically correct to WPSM's specifications.

There may be errors present in the input file, which are found through a series of checks. One may encounter a Lexer error, which occurs if there is an attempt to use a character that is invalid and unsupported in the grammar. If this occurs, the process is aborted and the user must reinitiate the compilation process with the corrected input file. If there are no Lexer errors, the next

checks that take place are the Parser checks. If there is an invalid token, or tokens are out of order, the Parser will throw an exception and will abort the process. Once the user fixes this error, they may recompile the input file and hopefully will have fixed the error. If the input file passes each of these tests, it is syntactically viable and an AST should be generated.

The grammar consists of several tokens that make up several different types of expressions with different functionalities. The details of the grammar are provided in the appendix, and one can determine the exact format from that detail. The structure of the AST is also defined in the grammar (in terms of root nodes and children). Each node on the tree represents a node specified in the WPSM grammar. Once the tree is generated without any errors, the Front-End Subsystem completes and sends its output to the Back-End Subsystem.

Note: Please see Appendix B. for the Front-end parser and lexer grammar

6.8 Backend-end subsystem

One could think of the Back-End Subsystem as the intermediate between the other two subsystems, the Front-End and the Library. Broadly, the Back-End begins with the AST, which has been handed to it by the Front-End, and ends at generated intermediate code (in our case JAVA) through interaction with the library. The generated JAVA code then interacts with the System.out and with the output and input files for WPSM, and is compiled by JAVA's own compiler. Now we will go through the subsystem in a little bit more depth.

When given the AST, the Back-End begins to traverse through the AST checking for symantic integrity. Though a piece of code could be syntactically correct, the pieces may not make sense. For example, an IF clause consists of IF (*conditional) statement* ELSE *statement*. If the conditional is accessing a variable that has not been initialized, and has no value, there is a symantic error. In the WPSM Back-End Subsystem, there are two walks of the AST by antlr's TreeWalker class. The primary walk determines which variables have been initialized and which have been used prior to initialization. When a variable is seen in the input file, that variable is appended to the Symbol Table, which keeps track of all variables in the program. If a variable is accessed, it must have been previously initialized, and must occur in the Symbol Table. Once this symbol table is created, we go on to the next step of the Back-End, code generation. In this step, the AST is walked for a second time and each type of token is analyzed and some generated JAVA code is outputted. The first step of code generation is to create the header to a JAVA file, and then to initialize all the variables that are contained in the Symbol Table. This is to ensure that the JAVA compiler will not encounter a variable that is not initialized. Once this takes place, each node of the AST is traversed, and from each node a little bit of code is created with the help of the Library. Once the entire AST is traversed, we end up with a well-formed JAVA file that will be compiled and run.

There are several symantic errors that are possible with the Back-End. Firstly, if a variable is used before it has been initialized, an exception will be thrown and the program will be aborted. A special case of errors is the Run-Time error. A Run-Time error is an error that occurs while a program is running. One such error is when a user attempts to multiply a string with an integer. The function that multiplies these two objects has a check to see what types of objects are being multiplies. If the check fails, and the two objects are not integers (as is the case), the program is aborted. Another such error is an attempt to read from an input file that does not exist. There are several other errors that are possible, but if one follows the specifications of WPSM, and pays heed to the error messages and warnings, they should be able to complete the compiling process without trouble.

Note: Please see Appendix B. and Appendix C. for the Back-end tree-walker grammar

6.9 WPSM library

In essence the muscle of our language resides in the library. It is responsible for making sure all that the user wants to do and all that the user is allowed to do with WPSM syntax is taken care of properly by the library. The WPSM library deals with the data input and is responsible for providing methods to the code generator for manipulating the data and outputting formatted XML output. A library was included in the WPSM architecture for many reasons. For one, it allowed for reusability of code. Since WPSM aims to deal mostly with large datafiles, most of the functions like READLINE() are called hundreds of times. A library helps simplify the intermediate code, allows for easy upgrades, and allows for easy debugging. Once the java code has been generated by the backend, it is a lot easier for one who may not know much about the complier to build on the language and make it even more powerful. Also, it can be integrated with quite possibly any language out there. All that would need to happen is that the language's back end would need to use WPSM library's functions. Another benefit is that changes in the syntax don't directly effect the library. The code generator deals with any grammar or syntax changes and makes sure to call the right library functions in the right order. The WPSM also comes with its own javadoc to help developers in upgrading and debugging.

Note: Please see Appendix C. for the WPSM Library source code.

7 Testing Plan

WPSM test plan is break down to several sections. Each section is focusing on specific task. The idea of this breaking down is able to provide a latest WPSM release after the each testing section and ensure that the development is continued.

WPSM source file testing
Only the file with extension ".xf" should be accepted by WPSM compiler. Everything else should not be accepted and the proper error message should be print on screen. For example, file not exists, file is not xf type, file doesn't have extension, etc. (Front-end developer should responsible for this handling)

WPSM syntax testing
All syntax that we support should pass WPSM parser, otherwise, syntax error. For example, all WPSM preserve key words should be compiled correctly. If the WPSM programmer uses the wrong key word, the proper error message should print out on the screen. (Front-end developer should responsible for this handling and please send out the latest WPSM syntax to me)

WPSM dependencies testing
Syntax dependence is a unique feature in WPSM that helps the WPSM programmer to deal with required XML field.

Note: Please see Appendix D. for detail dependence test plan.

There are dependencies on the WPSM keywords. For example, FN need to have value before anything runs, Attributes (RA) need to be initialized before value can begin to be set up, etc. Please verify these dependencies. (Both front-end and back-end should be responsible on this) This will be part of the syntax test case.

WPSM features testing
Includes basic and advance features. All the features we address in the reference manual should work properly and should be tested without error. For example, attribute manipulating, filtering, etc. (Back-end developer should responsible for all the features working properly.) Only the features that approve by you will be put into the reference manual, otherwise, the feature will be addressed to be included in future release.

Filtering
Column manipulation
XML tags (element, attribute) manipulation
Data manipulation
Arithmetic
Error handling ability

Release control
Every time when you are finish one testing you can ask me to build a release. This way we can keep all the latest working code for specify level. For

example, when the testers done with the basic testing (this mean the front-end and back-end developers fix all the bugs for basic function), we can have a new release, and then go on to next testing. This way we can always have a working latest release to hand in to the professor.

Subtleties of our language:

Our language supports all characters and numbers in the English language. The user should stay away from using any foreign characters in the data input file as some characters result in java ArrayOutOfBounds exception. Also, XML does not allow the user to use numbers or strings starting with numbers as Attributes. In this version, WPSM allows the user to input anything when setting RA. And lastly, WPSM supports the use of escape characters in the data input, however, does not support it in .xf files.

Appendix A.

sample.xf

```
BEGIN;
//-------------------------------------------------------------------------
//Set Input File Name
//
//This is not necessary in order to have the language run properly.
//The default value for this is "in.txt".
//-------------------------------------------------------------------------
FN := "fam.txt";

//-------------------------------------------------------------------------
//Set Output File Name
//
//This is not necessary in order to have the language run properly.
//The default value for this is "out.xml".
//-------------------------------------------------------------------------
OFN := "fam.xml";

//-------------------------------------------------------------------------
//Set Delimeter
//
//This is not necessary in order to have the language run properly.
//The default value for this is ",".
//-------------------------------------------------------------------------
FS := ",";

//-------------------------------------------------------------------------
//Set RA
//
//There are many ways you can set RA:
//  1) You can read the first line of your input file for the attributes
//         ex:     RA := READLINE();
//  2) You manually input the attributes
//                 ex:     RA := "apple, peach, orange";
//         note: you must delimit the Sring with the same delimeter set
//                 above.
//  3) You can change a specific attribute
//         ex:     $1 := "bana";
//                 $1 += "na";
//  4) You can append to attribute Vector
//                 ex:     RA += "seema";
//  NOTE: Once you have created your first record, you may not change RA.
//  NOTE: If you try to access an RA value that has not been initialized,
//        you will get a runtime error.
//  NOTE: Thie first index of RA anf RV start with 0.
//-------------------------------------------------------------------------

RA := READLINE();
$2 := "birthday";
PRINT(RA);

//-------------------------------------------------------------------------
//Create Record
//
//This command outputs a record to the XML file.
//There are several ways to create a record.
//  1) You can read a line from the input and set it in two ways
//    (a)   ex:     CREATERECORD(READLINE());
//    (b)  ex:     RV := READLINE();
//                 CREATERECORD(RV);
//  2) You can input your own string
//    (a) ex:      CREATEDRECORD("mom, dad, daughter, son");
//    (b) ex:      RV := "100,200,300,400";
//                 CREATERECORD(RV);
//  3) You can read in a line and change it
//           (a)   ex:     RV := READLINE();
//                 RV[2] := "300";
//                 CREATERECORD(RV);
//    (b) ex:      $line := READLINE();
//                 $line += "four";
//                 CREATERECORD($line);
```

```
//    (c)   ex:     RV := READLINE();
//                  RV += "four";          //this appends to the RV Vector
//                  CREATERECORD(RV);
//
//  NOTE: You can only access current RV values.  RV is overwritten everytime a
//        new setRV(Vector values) method is called.
//-----------------------------------------------------------------------
RV := READLINE();
RV[2] := "Edwards";
PRINT(RV);
CREATERECORD(RV);

PRINT(RV);
CREATERECORD("mom,dad,daughter,son");

//-----------------------------------------------------------------------
//Closes the input and output files.
//-----------------------------------------------------------------------
PRINT("done");
END;
```

Appendix B.

wpsm_lex_par_tw1.g

```
header {
// Date:           March 07, 2003
// Class:          COMS W4115, Programming Language and Translator - Spring 2003
// Project:        WPSM programming language project.
// Created by:     Wai Wong, Miqdad Mohammed
// Modified:
// Group:          Wai, Peter, Miqdad, Seema
// Compile ANTLR source:  java antlr.Tool wpsmc.g
// wpsm_lex_par_tw1.g is ANTLR source file
// Compile java source:          javac *.java
// Run the compiler:             java Main < input_file
// input_file is WPSM
//
// syntax source file
}

// Class TW
class TW extends TreeParser;
options {k = 2;}

startRule:         #("BEGIN" (stmts)* "END");
stmts:                     stmt_single | if_stmt | while_stmt;


while_stmt:        #("WHILE" bool_expr stmt_list);


if_stmt:           #("IF" bool_expr stmt_list (elsetail)?);
elsetail:          (elseif_stmt)+ (else_stmt)?
                           | else_stmt;
elseif_stmt:       #("ELSEIF" bool_expr stmt_list);
else_stmt:         #("ELSE" stmt_list);
stmt_list:         #(LBRACE (stmts)+)
                           | stmt_single;


stmt_single:       #(INC_ASSIGN a:choice) | #(DEC_ASSIGN b:choice) | #(ASSIGN
c:choice) | assign_p1_m1
                   | create_rc_stmt | print_stmt | SEMI;
choice:            c:VAR {Functions.finsert(c.getText());} arith_expr | ATTR_IDX
arith_expr | key arith_expr;
assign_p1_m1:      #(a:PLUS_ONE b:VAR) {Functions.find(b.getText());}
                   | #(c:MINUS_ONE d:VAR) {Functions.find(d.getText());};

print_stmt:        print_fnc;
create_rc_stmt:    create_rc_fnc;

arith_expr:        #(PLUS arith_expr arith_expr)
                   |#(MINUS arith_expr arith_expr)
                   |#(MUL arith_expr arith_expr)
                   |atom;

atom:                      key | readline_fnc | INT | STR_CONST | ATTR_IDX | a:VAR
{Functions.find(a.getText());} | ra_size | rv_size;

bool_expr:         #(LPAREN bool_combo);
bool_combo:        #("AND" bool_combo bool_combo)
                           | #("OR" bool_combo bool_combo)
                           | #(EQU booleq)
                           | #(NEQU booleq)
                           | bool_int
                           | bool_expr;

booleq:            boolatom (boolatom | eof | STR_CONST | INT | ra_size | rv_size);
boolatom:          a:VAR {Functions.find(a.getText());} | ATTR_IDX | readline_fnc |
key;
bool_int:          #(a:GTHAN_EQU b:VAR (c:INT | ra_size | rv_size)
{Functions.find(b.getText());})
```

```
                        | #(d:GTHAN e:VAR (f:INT | ra_size | rv_size)
{Functions.find(e.getText());})
                        | #(g:LTHAN_EQU h:VAR (i:INT | ra_size | rv_size)
{Functions.find(h.getText());})
                        | #(j:LTHAN k:VAR (l:INT | ra_size | rv_size)
{Functions.find(k.getText());});

print_fnc:          #("PRINT" arith_expr);

create_rc_fnc:      #("CREATERECORD"arith_expr);

readline_fnc:               "READLINE";
eof:                        "EOF";
empty_str:          "NULL";



/// BEGIN OF CHANGED AND NEW GRAMMAR
    key
            :                       "FN" | "FS" | "RT" | "RN" | "RA" | "OFN" | rv_key;
    rv_key
            :               #("RV" (rv_assig)?);
    rv_assig
            :       #(LBRACKET INT );
/// END OF CHANGED AND NEW GRAMMAR

ra_size :  "RA_SIZE";

rv_size :  "RV_SIZE";

// Class P
////////////////////////////////////////////////////////////////////
class P extends Parser;
options {
    buildAST = true;
    k = 2;
    defaultErrorHandler = false;
}

// Program List (Block)
    startRule
            :                       "BEGIN"^ SEMI!
                                           (stmt_single | if_stmt | while_stmt)*
                                    "END" SEMI!;

// Statement List (Block)
    while_stmt
            :                       "WHILE"^ bool_expr stmt_list;
    if_stmt
            :                       "IF"^ bool_expr stmt_list (((elseif_stmt)+
(else_stmt)?) |
else_stmt)?;
    elseif_stmt
            :                       "ELSEIF"^ bool_expr stmt_list;
    else_stmt
            :                       "ELSE"^ stmt_list;
    stmt_list
            :                       (LBRACE^ (stmt_single | if_stmt | while_stmt)+
RBRACE!)
                                    | stmt_single;

// Assignment Statements
    stmt_single
            :                       assign_var | assign_p1_m1 | create_rc_stmt |
                                    print_stmt | SEMI;
/*                                  exception catch [RecognitionException ex] {
                                           System.out.println("error: illegal
variable assigment -" +
                                           " line " + ex.getLine() + "; column " +
ex.getColumn());
                                           System.out.println("Tips: $literal :=
[$[integer], integer, string
constant]");
                                    }      */

    assign_var
```

```
        :                       (VAR | ATTR_IDX | key) (INC_ASSIGN^ | DEC_ASSIGN^
| ASSIGN^)
(arith_expr) SEMI!;
    assign_p1_m1
        :                       VAR (PLUS_ONE^ | MINUS_ONE^) SEMI!;
    create_rc_stmt
        :                       create_rc_fnc SEMI!;
    print_stmt
        :                       print_fnc SEMI!;
    arith_expr
        :                       sum_expr;
//                              {System.out.println("arith_expr");};
    sum_expr
        :               prod_expr ((PLUS^ | MINUS^) prod_expr)*;
    prod_expr
        :               atom (MUL^ atom)*;
    atom
        :               key | readline_fnc | INT | STR_CONST | ATTR_IDX | VAR |
ra_size | rv_size |
                        (LPAREN! arith_expr RPAREN!);

// Comparison Statements
    bool_expr
        :               LPAREN^ bool_combo RPAREN!;
/*                              exception catch [RecognitionException ex] {
                                System.out.println("error: illegal
boolean expression -" +
                                " line " + ex.getLine() + "; column " +
ex.getColumn());
                                System.out.println("Tips: [IF (expr),
ELSEIF (expr), WHILE (expr)]");
                        }       */
    bool_combo
        :               bool_single (("AND"^ | "OR"^) bool_single)*;
    bool_single
        :               bool_var | bool_int | bool_expr;
    bool_var
        :               bool_arg (EQU^ | NEQU^) (bool_arg | STR_CONST | INT |
eof | ra_size | rv_size);
    bool_arg
        :               readline_fnc | ATTR_IDX | VAR | key;
/*  bool_var
        :               VAR (EQU^ | NEQU^) (STR_CONST | ATTR_IDX | INT);
    bool_attr
        :               ATTR_IDX (EQU^ | NEQU^) (STR_CONST | ATTR_IDX | VAR);
*/
    bool_int
        :               VAR (GTHAN_EQU^ | GTHAN^ | LTHAN_EQU^ | LTHAN^) (INT |
ra_size | rv_size);
/*  bool_readline
        :               readline_fnc (EQU^ | NEQU^) (eof | STR_CONST);
*/

// Build-in Function;
    print_fnc
        :               "PRINT"^ LPAREN! arith_expr RPAREN!;
    create_rc_fnc
        :               "CREATERECORD"^ LPAREN! arith_expr RPAREN!;

// BEGIN OF CHANGED GRAMMAR
    readline_fnc
        :               "READLINE"^ LPAREN! RPAREN!;
// END OF CHANGED GRAMMAR

    eof
        :               "EOF";
    empty_str
        :               "NULL";

/// BEGIN OF CHANGED AND NEW GRAMMAR
    key
        :                       "FN" | "FS" | "RT" | "RN" | "RA" | "RV" | "OFN" |
rv_key;
    rv_key
        :                       "RV"^ array_elem;
    array_elem
        :                       LBRACKET^ INT RBRACKET!;
```

```
/// END OF CHANGED AND NEW GRAMMAR

    ra_size
            :                       "RA_SIZE";
    rv_size
            :                       "RV_SIZE";


// Class L
////////////////////////////////////////////////////////////////////
class L extends Lexer;
options {
    k = 2;
    charVocabulary = '\3'..'\377';
}
// Identifier rules
    protected DIGIT
            :                       ('0'..'9');
    protected DQUOTES
            :                       '"';
    LITERAL
            :                       ('a'..'z'|'A'..'Z')
('a'..'z'|'A'..'Z'|'0'..'9'|'_')*;       // Can't make LITERAL protected?
    INT
            :                       (DIGIT)+;
    ATTR_IDX
            :                       ('$' INT);
    VAR
            :                       ('$' LITERAL);
    STR_CONST
            :                       DQUOTES (' ' | '!' | '#'..'~')* DQUOTES;

// Assignment operators
    ASSIGN
            :                       ":=";
    INC_ASSIGN
            :                       "+=";
    DEC_ASSIGN
            :                       "-=";
    PLUS_ONE
            :                       "++";
    MINUS_ONE
            :                       "--";

// Arithematic operators
    PLUS
            :                       '+';
    MINUS
            :                       '-';
    MUL
            :                       '*';
// Comparing operators
    EQU
            :                       "==";
    NEQU
            :                       "!=";
    GTHAN_EQU
            :                       ">=";
    LTHAN_EQU
            :                       "<=";
    GTHAN
            :                       '>';
    LTHAN
            :                       '<';
// Block terminators
    SEMI
            :                       ';';
    COMMA
            :                       ',';
    LPAREN
            :                       '(';
    RPAREN
            :                       ')';
    LBRACE
            :                       '{';
    RBRACE
            :                       '}';
```

```
    LBRACKET
            :                       '[';
    RBRACKET
            :                       ']';

// Skipped characters
    NEWLINE
            :                       ("\r\n" | '\n' | "\r\t")              // DOS or
Unix; "\r\t" are used in some

            // editor program as next line character.
                                    {
                                            newline();
                                            $setType(Token.SKIP);
                                    };
    COMMENT
            :                       ("//") (' '..'~' | '\t')* NEWLINE
                                    {
                                            $setType(Token.SKIP);
                                    };
    WS
            :                       (' ' | '\t')
                                    {
                                            $setType(Token.SKIP);
                                    };
```

Appendix C.

wpsm_lex_par_tw2.g

```
// Date:           March 07, 2003
// Class:          COMS W4115, Programming Language and Translator - Spring 2003
// Project:        WPSM programming language project.
// Created by:     Miqdad Mohammed
// Modified:
// Group:          Wai, Peter, Miqdad, Seema
// Compile ANTLR source:  java antlr.Tool wpsmc.g
// wpsm_lex_par_tw2.g is ANTLR source file
// Compile java source:           javac *.java
// Run the compiler:              java Main < input_file
// input_file is WPSM
//
// Class TW2
class TW2 extends TreeParser;
options {k = 2;}

startRule:         {Functions.test = false;}#("BEGIN" {Functions.println("public
class " + Functions.classname + "\n{\n\tpublic static void main(String []
args)\n\t{"); Functions.outprintLL();} ({Functions.printtab();} stmts)* "END")
{Functions.printtab();
Functions.print("XLib.closeFile();");Functions.println("\t}\n}");
Functions.close();};
stmts:                     stmt_single {Functions.println(";");} | if_stmt |
while_stmt;


while_stmt:        #("WHILE" {Functions.print("while");} bool_expr
{Functions.print("\n");} stmt_list);


if_stmt:           #("IF" {Functions.print("if");} bool_expr
{Functions.print("\n");} stmt_list (elsetail)?);
elsetail:          (elseif_stmt)+ (else_stmt)?
                           | else_stmt;
elseif_stmt:       #("ELSEIF" {Functions.printtab(); Functions.print("else if");}
bool_expr {Functions.print("\n");;} stmt_list);
else_stmt:         #("ELSE" {Functions.printtab(); Functions.print("else");}
{Functions.print("\n");;} stmt_list);
stmt_list:         #(LBRACE {Functions.printtab(); Functions.println("{ ");
Functions.tab++;} ({Functions.printtab();} stmts)+ {Functions.tab--;
Functions.printtab(); Functions.println("}");})
                           | {Functions.tab++; Functions.printtab(); Functions.tab-
-;} stmt_single {Functions.println(";");};


stmt_single:       #(INC_ASSIGN a:incchoice) | #(DEC_ASSIGN decchoice) | #(ASSIGN
assignchoice) | assign_pl_m1
                   | create_rc_stmt | print_stmt | SEMI;
incchoice:         a:VAR {Functions.print(Functions.conv(a.getText()) + ".inc(");}
arith_expr {Functions.print(")");}
                   | b:ATTR_IDX {Functions.print("XLib.setRA(XLib.getRA(" +
Functions.index(b.getText()) + ") + ");} arith_expr {Functions.print(", " +
Functions.index(b.getText()) + ")");}
                   | c:key1 {Functions.print("XLib.set" + c.getText() + "(XLib.get"
+ c.getText() + "() + ");} arith_expr {Functions.print(")");}
                  | d:key2 {Functions.print("XLib.setRA(XLib.toString(XLib.getRA())
+ ");} arith_expr {Functions.print(")");}
                   | e:rv_key
{if(Functions.rv){Functions.print("XLib.setRV(XLib.getRV("+ Functions.rvint + ") +
");}else{Functions.print("XLib.setRV(XLib.toVector(XLib.toString(XLib.getRV()) +
");}} arith_expr {if(Functions.rv){Functions.print(", " + Functions.rvint + ")");
Functions.rv = false;}else{Functions.print("))");}};
decchoice:         a:VAR {Functions.print(Functions.conv(a.getText()) + ".dec(");}
arith_expr {Functions.print(")");}
                   | b:ATTR_IDX {System.out.println("WPSM exception: attempting to
decrement " + b.getText() + "\nSorry! error(s) is/are found, WPSM compiling
process aborted."); System.exit(0);} arith_expr
                   | c:key1 {System.out.println("WPSM exception: attempting to
decrement " + c.getText() + "\nSorry! error(s) is/are found, WPSM compiling
process aborted."); System.exit(0);}arith_expr
```

```
                  | d:key2 {System.out.println("WPSM exception: attempting to
decrement " + d.getText() + "\nSorry! error(s) is/are found, WPSM compiling
process aborted."); System.exit(0);}arith_expr
                  | e:rv_key {System.out.println("WPSM exception: attempting to
decrement " + e.getText() + "\nSorry! error(s) is/are found, WPSM compiling
process aborted."); System.exit(0);}arith_expr;
assignchoice:             a:VAR {Functions.print(Functions.conv(a.getText()) +
".replaceVal(");} arith_expr {Functions.print(")");}
                  | b:ATTR_IDX {Functions.print("XLib.setRA(\"\" + (");}
arith_expr {Functions.print("), " + Functions.index(b.getText()) + ")");}
                  | c:key1 {Functions.print("XLib.set" + c.getText() + "(\"\" +
(");} arith_expr {Functions.print("))");}
                  | d:key2 {Functions.print("XLib.set" + d.getText() +
"(XLib.toVector(\"\" + (");} arith_expr {Functions.print(")))");}
                  | e:rv_key
{if(Functions.rv){Functions.print("XLib.setRV(");}else{Functions.print("XLib.setRV
(XLib.toVector(");}} arith_expr {if(Functions.rv){Functions.print(", " +
Functions.rvint + ")"); Functions.rv = false;}else{Functions.print("))");}};

assign_p1_m1:     #(a:PLUS_ONE b:VAR) {Functions.print(Functions.conv(b.getText())
+ ".inc(\"1\")");}
                        | (c:MINUS_ONE d:VAR)
{Functions.print(Functions.conv(d.getText()) + ".dec(\"1\")");};

print_stmt:       print_fnc;
create_rc_stmt:   create_rc_fnc;

arith_expr:       #(PLUS {Functions.print("Functions.add(");} arith_expr
{Functions.print(",");} arith_expr {Functions.print(")");})
                  |#(MINUS {Functions.print("Functions.sub(");} arith_expr
{Functions.print(",");} arith_expr {Functions.print(")");})
                  |#(MUL {Functions.print("Functions.mul(");} arith_expr
{Functions.print(",");} arith_expr {Functions.print(")");})
                  |atom;

atom:                     a:key1 {Functions.print("XLib.get" + a.getText() +
"()");} | b:key2 {Functions.print("XLib.toString(XLib.getRA())");}
                  | rv_key {if(Functions.rv){Functions.print("XLib.getRV("+
Functions.rvint + ")");Functions.rv =
false;}else{Functions.print("XLib.toString(XLib.getRV())");}}
                  | readline_fnc | c:INT {Functions.print("\"" + c.getText() +
"\"");}| d:STR_CONST {Functions.print(d.getText());}
                  | f:ATTR_IDX {Functions.print("XLib.getRA(" +
Functions.index(f.getText()) + ")");}| e:VAR
{Functions.print(Functions.conv(e.getText()) + ".getstringVal()");}
                  | ra_size {Functions.print("String.valueOf(XLib.getRASize())");}
                 | rv_size {Functions.print("String.valueOf(XLib.getRVSize())");};


bool_expr:        #(LPAREN {Functions.print("(");} bool_combo
{Functions.print(")");});
bool_combo:       #("AND" bool_combo {Functions.print(" && ");} bool_combo)
                        | #("OR" bool_combo {Functions.print(" || ");}
bool_combo)
                        | #(EQU boolatom1 boolatom2)
                        | #(NEQU {Functions.print("!(");} boolatom1 boolatom2
{Functions.print(")");})
                        | bool_int
                        | bool_expr;

boolatom1:        a:VAR {Functions.print(Functions.conv(a.getText()) +
".getstringVal()");}
                  | b:ATTR_IDX {Functions.print("XLib.getRA(" +
Functions.index(b.getText()) + ")");}
                  | readline_fnc
                  | c:key {if(Functions.rv){Functions.print("XLib.getRV(" +
Functions.rvint + ")"); Functions.rv = false;}
                           else if(c.getText().equals("RA") ||
c.getText().equals("RV")){Functions.print("XLib.toString(XLib.get" + c.getText() +
"())");}
                           else{Functions.print("XLib.get" + c.getText() +
"()");}};
boolatom2:        a:VAR {Functions.print(".equals(" + Functions.conv(a.getText())
+ ".getstringVal())");}
                  | b:ATTR_IDX {Functions.print("XLib.getRA(" +
Functions.index(b.getText()) + ")");}
```

```
                  | {Functions.print(".equals(");} readline_fnc
{Functions.print(")");}
                  | c:key {if(Functions.rv){Functions.print(".equals(XLib.getRV("
+ Functions.rvint + "))"); Functions.rv = false;}
                           else if(c.getText().equals("RA") ||
c.getText().equals("RV")){Functions.print(".equals(XLib.toString(XLib.get" +
c.getText() + "()))");}
                           else{Functions.print(".equals(XLib.get" + c.getText() +
"())");}}
                  | eof {Functions.print(".equals(\"EOF\")");}
                  | d:STR_CONST {Functions.print(".equals(" + d.getText() + ")");}
                  | e:INT {Functions.print(".equals(\"" + e.getText() + "\")");}
                  | ra_size
{Functions.print(".equals(String.valueOf(XLib.getRASize()))");}
                  | rv_size
{Functions.print(".equals(String.valueOf(XLib.getRVSize()))");};


bool_int:         #(a:GTHAN_EQU b:VAR {Functions.print(Functions.conv(b.getText())
+ ".gte(");} (c:INT {Functions.print(c.getText() + ")");} | rv_size
{Functions.print("XLib.getRVSize())");}| ra_size
{Functions.print("XLib.getRASize())");}))
                          | #(d:GTHAN e:VAR
{Functions.print(Functions.conv(e.getText()) + ".gt(");} (f:INT
{Functions.print(f.getText() + ")");} | rv_size
{Functions.print("XLib.getRVSize())");}| ra_size
{Functions.print("XLib.getRASize())");}))
                          | #(g:LTHAN_EQU h:VAR
{Functions.print(Functions.conv(h.getText()) + ".lte(");} (i:INT
{Functions.print(i.getText() + ")");} | rv_size
{Functions.print("XLib.getRVSize())");}| ra_size
{Functions.print("XLib.getRASize())");}))
                          | #(j:LTHAN k:VAR
{Functions.print(Functions.conv(k.getText()) + ".lt(");} (l:INT
{Functions.print(l.getText() + ")");} | rv_size
{Functions.print("XLib.getRVSize())");}| ra_size
{Functions.print("XLib.getRASize())");}));


print_fnc:        #("PRINT" {Functions.print("System.out.println(");}
arith_expr){Functions.print(")");};


create_rc_fnc:    #("CREATERECORD"
{Functions.print("XLib.createRecord(XLib.toVector(");} arith_expr
{Functions.print("))");});
/*{Functions.rec = true; Functions.print("XLib.createRecord(");}
                                        (b:key1 {Functions.print("XLib.get" +
b.getText() + "()");}
                                        | c:key2 {Functions.print("XLib.get" +
c.getText() + "()");}
                                        | rv_key
{if(Functions.rv){Functions.print("XLib.toVector(XLib.getRV("+ Functions.rvint +
"))");Functions.rv = false;}else{Functions.print("XLib.getRV()");}}
                                        | a:STR_CONST
{Functions.print("XLib.toVector(" + a.getText() + ")");}
                                        | {Functions.test = true;
Functions.print("XLib.readToVector()");} readline_fnc {Functions.test = false;}
                                        | d:VAR
{Functions.print(Functions.conv(d.getText()) + ".getstringVal()");}))
                                        {Functions.print(")");};
*/
readline_fnc:     "READLINE"
{if(!Functions.test){Functions.print("XLib.readLine()");};};

eof:                    "EOF";
empty_str:        "NULL";
key:                    key1 | key2 | rv_key;
key1:                   "FN" | "FS" | "RT" | "RN" | "OFN";
key2:                   "RA";

rv_key:           #("RV" (rv_assig)?);
rv_assig:  #(LBRACKET {Functions.rv = true;} d:INT {Functions.rvint =
Integer.parseInt(d.getText());});

ra_size:   "RA_SIZE";
rv_size:   "RV_SIZE";
```

Main.java

```
import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;

class Main {
    public static void main(String[] args) {
            String temp = "";
            try {
                    temp = args[0];
                    Functions.output = temp;
                    Functions.initial();
                    FileReader theFile = new FileReader(temp);
                    BufferedReader input = new BufferedReader(theFile);
                    //DataInputStream input = new DataInputStream(temp);
                    //System.out.print(input);
                L lexer = new L(input);

                    P parser = new P(lexer);
                    parser.startRule();

                    CommonAST parseTree = (CommonAST)parser.getAST();
//                  System.out.println('\n' + parseTree.toStringList());

//                  System.out.println('\n' + "Displaying WPSM AST...");
//                  ASTFrame frame = new ASTFrame("WPSM AST", parseTree);
//                  frame.setVisible(true);
                    System.out.println("first walk");
                    TW walker = new TW();
                    walker.startRule(parseTree);


//                  L2 lexer2 = new L2(input);
                    System.out.println("second walk");
//                  P2 parser2 = new P2(lexer);
//                  parser2.startRule();

                    theFile = new FileReader(temp);
                    input = new BufferedReader(theFile);
                    //DataInputStream input = new DataInputStream(temp);
                    //System.out.print(input);
            L2 lexer2 = new L2(input);

                    P2 parser2 = new P2(lexer2);
                    parser2.startRule();

                    CommonAST parseTree2 = (CommonAST)parser2.getAST();

                    TW2 walker2 = new TW2();
                    walker2.startRule(parseTree2);

            }catch(FileNotFoundException e1) {
                    System.err.println("WPSM exception: the file " + temp + " was
not found\nSorry! error(s) is/are found, WPSM compiling process aborted.");
        }catch(Exception e) {
                    System.err.println("WPSM exception: "+e);
                    System.err.println("Sorry! error(s) is/are found, WPSM compiling
process aborted.");
        }
    }
}
```

LinkedList.java

```
// LinkedList class
//
// CONSTRUCTION: with no initializer
// Access is via LinkedListItr class
//
```

```
// *****************PUBLIC OPERATIONS*******************
// boolean isEmpty( )     --> Return true if empty; else false
// void makeEmpty( )      --> Remove all items
// LinkedListItr zeroth( )--> Return position to prior to first
// LinkedListItr first( ) --> Return first position
// void insert( x, p )    --> Insert x after current iterator position p
// void remove( x )       --> Remove x
// LinkedListItr find( x )
//                        --> Return position that views x
// LinkedListItr findPrevious( x )
//                        --> Return position prior to x
// *****************ERRORS******************************
// No special errors

/**
 * Linked list implementation of the list
 *    using a header node.
 * Access to the list is via LinkedListItr.
 * @author Mark Allen Weiss
 * @see LinkedListItr
 */
public class LinkedList
{
    /**
     * Construct the list
     */
    public LinkedList( )
    {
        header = new ListNode( null );
    }

    /**
     * Test if the list is logically empty.
     * @return true if empty, false otherwise.
     */
    public boolean isEmpty( )
    {
        return header.next == null;
    }

    /**
     * Make the list logically empty.
     */
    public void makeEmpty( )
    {
        header.next = null;
    }


    /**
     * Return an iterator representing the header node.
     */
    public LinkedListItr zeroth( )
    {
        return new LinkedListItr( header );
    }

    /**
     * Return an iterator representing the first node in the list.
     * This operation is valid for empty lists.
     */
    public LinkedListItr first( )
    {
        return new LinkedListItr( header.next );
    }

    /**
     * Insert after p.
     * @param x the item to insert.
     * @param p the position prior to the newly inserted item.
     */
    public void insert( Variable x, LinkedListItr p )
    {
        if( p != null && p.current != null )
            p.current.next = new ListNode( x, p.current.next );
    }
```

```
        /**
         * Return iterator corresponding to the first node containing an item.
         * @param x the item to search for.
         * @return an iterator; iterator isPastEnd if item is not found.
         */
        public LinkedListItr find( Variable x )
        {
/* 1*/      ListNode itr = header.next;

/* 2*/      while( itr != null && !itr.element.equals( x ) )
/* 3*/          itr = itr.next;

/* 4*/      return new LinkedListItr( itr );
        }

        /**
         * Return iterator prior to the first node containing an item.
         * @param x the item to search for.
         * @return appropriate iterator if the item is found. Otherwise, the
         * iterator corresponding to the last element in the list is returned.
         */
        public LinkedListItr findPrevious( Variable x )
        {
/* 1*/      ListNode itr = header;

/* 2*/      while( itr.next != null && !itr.next.element.equals( x ) )
/* 3*/          itr = itr.next;

/* 4*/      return new LinkedListItr( itr );
        }

        /**
         * Remove the first occurrence of an item.
         * @param x the item to remove.
         */
        public void remove( Variable x )
        {
            LinkedListItr p = findPrevious( x );

            if( p.current.next != null )
                p.current.next = p.current.next.next;  // Bypass deleted node
        }

        public void change( Variable x, LinkedListItr itr )  //changes the current
number heald in the exp class
        {
            itr.current.element = x;
        }

               public static void opLL(LinkedList theList)
               {
                       if( theList.isEmpty( ) )
                       {}
            else
            {
                LinkedListItr itr = theList.first( );   //this print method will print
an (x * 10^y) format
                for( ; !itr.isPastEnd( ); itr.advance( ) )
                {
                                    if(itr.retrieve().toString().charAt(0) == '$')
                                            Functions.println("\tVariable " +
Functions.conv(itr.retrieve().toString()) + " = new Variable(" + "\"" +
Functions.conv(itr.retrieve().toString()) + "\"" + ", \"\");");
                                    else
                                            Functions.println("\tString " +
itr.retrieve() + ";");
                }
            }

            System.out.println( );

               }

        // Simple print method
        public static void printList( LinkedList theList )
        {
            if( theList.isEmpty( ) )
```

```
                System.out.print( "Empty list" );
            else
            {
                LinkedListItr itr = theList.first( );   //this print method will print
an (x * 10^y) format
                for( ; !itr.isPastEnd( ); itr.advance( ) )
                {
                                        if(itr.retrieve().toString().charAt(0) == '$')
                                                System.out.println(itr.retrieve() + " = "
+ itr.retrieve().getstringVal());
                                        else
                                                System.out.println("built in var = " +
itr.retrieve());
                }
            }

            System.out.println( );
        }

        private ListNode header;
    }
```

LinkedListItr.java

```
    // LinkedListItr class; maintains "current position"
    //
    // CONSTRUCTION: Package friendly only, with a ListNode
    //
    // ******************PUBLIC OPERATIONS*********************
    // void advance( )        --> Advance
    // boolean isPastEnd( )   --> True if at "null" position in list
    // Object retrieve        --> Return item in current position

    /**
     * Linked list implementation of the list iterator
     *    using a header node.
     * @author Mark Allen Weiss
     * @see LinkedList
     */
    public class LinkedListItr
    {
        /**
         * Construct the list iterator
         * @param theNode any node in the linked list.
         */
        LinkedListItr( ListNode theNode )
        {
            current = theNode;
        }

        /**
         * Test if the current position is past the end of the list.
         * @return true if the current position is null.
         */
        public boolean isPastEnd( )
        {
            return current == null;
        }

        /**
         * Return the item stored in the current position.
         * @return the stored item or null if the current position
         * is not in the list.
         */
        public Variable retrieve( )
        {
            return isPastEnd( ) ? null : current.element;
        }

        /**
         * Advance the current position to the next node in the list.
         * If the current position is null, then do nothing.
         */
```

```
        public void advance( )
        {
            if( !isPastEnd( ) )
                current = current.next;
        }

        ListNode current;    // Current position
    }
```

ListNode.java

```
    // Basic node stored in a linked list
    // Note that this class is not accessible outside
    // of package DataStructures

    class ListNode
    {
            // Constructors
        ListNode( Variable theElement ) //contains an element of the class exp
        {
            this( theElement, null );
        }

        ListNode( Variable theElement, ListNode n )
        {
            element = theElement;
            next    = n;
        }

            // Friendly data; accessible by other package routines
        Variable element;
        ListNode next;
    }
```

Functions.java

```
import java.io.*;

public class Functions
{
        static LinkedList templl = new LinkedList();
        static LinkedListItr itr = templl.zeroth();
        static String output;
        static String classname;
        public static FileWriter writer;
        public static BufferedWriter bu;
        public static PrintWriter fileOut;
        public static boolean test = false;
        public static boolean rec = false;
        public static boolean rv = false;
        public static boolean boolvar = false;
        public static int rvint;
        public static int tab = 1;

        public static void initial()
        {       boolean ex = false;
                try{
                        String temp = "";
                        String ext = "";
                        for(int i = 0; i < output.length(); i++)
                        {
                                if(output.charAt(i) == '.')
                                        ex = true;
                                if(!ex)
                                        temp = temp + output.charAt(i);
                                else
                                        ext = ext + output.charAt(i);
                        }
                        if(ext.equals(".xf"))
                        {
```

```
                                classname = temp;
                                temp = temp + ".java";
                                output = temp;
                                writer = new FileWriter(output);
                                bu = new BufferedWriter(writer);
                                fileOut = new PrintWriter(bu);
                        }
                        else
                        {
                                System.out.println("WPSM exception: Input File in
incorrect format.  WPSM only accepts .xf files\nSorry! error(s) is/are found, WPSM
compiling process aborted.");
                                System.exit(0);
                        }
                }
                catch(IOException e2)
                {
                        System.out.println("WPSM exception: there was an error with the
output file\nSorry! error(s) is/are found, WPSM compiling process aborted.");
                }
        }

        public static void insert(String x, String value)
        {
                Variable temp = new Variable(x, value);
                templl.insert(temp, itr);
                itr.advance();
        }

        public static void printLL()
        {
                LinkedList.printList(templl);
        }

        public static void outprintLL()
        {
                LinkedList.opLL(templl);
        }

        public static Variable find(String x)
        {
                Variable temp = new Variable(x);
                itr = templl.find(temp);
                if(!(itr.retrieve() == null))
                {
                }
                else
                {
                        System.out.println("WPSM exception: The variable " + x + " that
you are trying to access has not been initialized\nSorry! error(s) is/are found, WPSM
compiling process aborted.");
                        System.exit(0);
                }

                return itr.retrieve();
        }

        public static void finsert(String x)
        {
                finsert(x, "");
        }

        public static Variable finsert(String x, String value)
        {
                LinkedListItr itr2 = itr;
                Variable temp = new Variable(x, value);
                itr2 = templl.find(temp);
                if(!(itr2.retrieve() == null))
                {
                        itr2.retrieve().replaceVal(value);
                }
                else
                {
                        insert(x, value);
                }
                return itr2.retrieve();
        }
```

```
public static int index(String ind)
{
	int index;
	int temp = ind.length();
	char[] tempchar = new char[temp - 1];

	for(int i = 1; i < temp; i++)
	{
		int temp2 = i-1;
		tempchar[temp2] = ind.charAt(i);
	}
	index = Integer.parseInt(String.valueOf(tempchar));
	return index;
}

public static String conv(String str)
{
	int temp = str.length();
	char[] tempchar = new char[temp-1];

	for(int i = 1; i < temp; i++)
	{
		int temp2 = i-1;
		tempchar[temp2] = str.charAt(i);
	}
	return String.valueOf(tempchar);
}

public static void println(String output)
{
	fileOut.println(output);
}

public static void print(String output)
{
	fileOut.print(output);
}

public static void close()
{
	fileOut.close();
}

public static void printtab()
{
	for(int i = 0; i < tab; i++)
	{
		print("\t");
	}
}

public static String add(String x, String y)
{
	if(x.equals(new String("")) || y.equals(new String("")))
	{
		return x + y;
	}
	else if(isInt(x) && isInt(y))
	{
		int temp = Integer.parseInt(x) + Integer.parseInt(y);
		return temp + "";
	}
	else
	{
		return x + y;
	}
}

public static String sub(String x, String y)
{
	if(isInt(x) && isInt(y))
	{
		int temp = Integer.parseInt(x) - Integer.parseInt(y);
		return temp + "";
	}
	else if(isInt(x))
```

```
		{
			System.out.println("WPSM exception: Trying to subtract the
string " + y + " from an integer\nSorry! error(s) is/are found, WPSM compiling process
aborted.");
			System.exit(0);
			return "";
		}
		else if(isInt(y))
		{
			System.out.println("WPSM exception: Trying to subtract an
integer from the string " + x + "\nSorry! error(s) is/are found, WPSM compiling
process aborted.");
			System.exit(0);
			return "";
		}
		else
		{
			System.out.println("WPSM exception: Trying to subtract the
string " + y + " from the string " + x + "\nSorry! error(s) is/are found, WPSM
compiling process aborted.");
			System.exit(0);
			return "";
		}
	}

	public static String mul(String x, String y)
	{
		if(isInt(x) && isInt(y))
		{
			int temp = Integer.parseInt(x) * Integer.parseInt(y);
			return temp + "";
		}
		else if(isInt(x))
		{
			System.out.println("WPSM exception: Trying to multiply the
string " + y + " with an integer\nSorry! error(s) is/are found, WPSM compiling process
aborted.");
			System.exit(0);
			return "";
		}
		else if(isInt(y))
		{
			System.out.println("WPSM exception: Trying to multiply an
integer with the string " + x + "\nSorry! error(s) is/are found, WPSM compiling
process aborted.");
			System.exit(0);
			return "";
		}
		else
		{
			System.out.println("WPSM exception: Trying to multiply the
string " + y + " with the string " + x + "\nSorry! error(s) is/are found, WPSM
compiling process aborted.");
			System.exit(0);
			return "";
		}
	}

	public static boolean isInt(String x)
	{
		try
		{
			Integer.parseInt(x);
			return true;
		}
		catch(Exception e)
		{
			return false;
		}
	}
}
```

Variable.java

```
public class Variable
{
	String name;
	String value;

	public Variable(String nam)
	{
		name = nam;
	}

	public Variable(String nam, String val)
	{
		name = nam;
		value = val;
	}

	public boolean isInt()
	{
		try
		{
			Integer.parseInt(value);
			return true;
		}
		catch(Exception e)
		{
			return false;
		}
	}

	public int getintVal()
	{
		return Integer.parseInt(value);
	}

	public String getstringVal()
	{
		return value;
	}

	public String toString()
	{
		return name;
	}

	public boolean equals(Variable x)
	{
		return name.equals(x.name);
	}

	public void changeInt(int x)
	{
		int temp = Integer.parseInt(value);
		temp += x;
		value = String.valueOf(temp);
	}

	public void replaceVal(String val)
	{
		value = val;
	}

	public void replaceVal(int val)
	{
		value = "" + val;
	}

	public boolean valequals(String s)
	{
		if(value.equals(s))
			return true;
		else
			return false;
	}
```

```
	public boolean lt(int x)
	{
		if(isInt())
		{
			if(getintVal()< x)
				return true;
			else
				return false;
		}
		else
		{
			System.out.println(name + " is not an integer value, and cannot
be compared to int " + x);
			System.exit(0);
			return false;
		}
	}

	public boolean gt(int x)
	{
		if(isInt())
		{
			if(getintVal()> x)
				return true;
			else
				return false;
		}
		else
		{
			System.out.println(name + " is not an integer value, and cannot
be compared to int " + x);
			System.exit(0);
			return false;
		}
	}

	public boolean lte(int x)
	{
		if(isInt())
		{
			if(getintVal()<= x)
				return true;
			else
				return false;
		}
		else
		{
			System.out.println(name + " is not an integer value, and cannot
be compared to int " + x);
			System.exit(0);
			return false;
		}
	}

	public boolean gte(int x)
	{
		if(isInt())
		{
			if(getintVal()>= x)
				return true;
			else
				return false;
		}
		else
		{
			System.out.println(name + " is not an integer value, and cannot
be compared to int " + x);
			System.exit(0);
			return false;
		}
	}

	public void dec(String x)
	{
		if(isInt())
		{
			if(Functions.isInt(x))
```

```
                                value = String.valueOf(getintVal() -
Integer.parseInt(x));
                        else
                        {
                                System.out.println("WPSM exception: attempting to
subtract a string from int $" + name + "\nSorry! error(s) is/are found, WPSM compiling
process aborted.");
                                System.exit(0);
                        }
                }
                else
                {
                        System.out.println("WPSM exception: $" + name + " is not an
integer value and cannot be decremented\nSorry! error(s) is/are found, WPSM compiling
process aborted.");
                        System.exit(0);
                }
        }

        public void inc(String x)
        {
                if(isInt())
                {
                        if(Functions.isInt(x))
                                value = String.valueOf(getintVal() +
Integer.parseInt(x));
                        else
                                value = value + x;
                }
                else
                {
                        value = value + x;
                }
        }
}
```

XLib.java

```
//*************************************************************************
// XML_lib.java       Author: Seema Gupta
//
// Library for WPSM
// Creates XML source code file.
//*************************************************************************
import java.util.StringTokenizer;
import java.io.*;
import java.util.Vector;

public class XLib{
    //bufferd reader, writer and printer
    public static FileReader fr;
    public static FileWriter fw;
    public static BufferedReader inFile;
    public static BufferedWriter bw;
    public static PrintWriter outFile;
    public static StringTokenizer tokenizer;

    //WPSM variables
    public static String in_File = "in.txt";         //input file name
    public static String out_File = "out.xml";              //output file name
    public static String delimeter = ",";    //delimeter
    public static String RT = "RT";          //root name
    public static String RN = "RN";   //element name
    public static Vector RA = new Vector();  //attribute vector
    public static Vector RV = new Vector();  //record vector
    public static String prolog = "<?xml version=\"1.0\"?>";        //prolog
    public static String comments = "";      //comments
    public static String line = "";   //empty line
    public static int count = 0;      //count variable
    public static boolean isCreated = false; //openFile()
    public static boolean isCreated2 = false;        //created first record
    public static File inF, outF;
```

```
    /**
     * Check to see if the input file specified by the user exists.
     */
    public static void setFN(){
       setFN(in_File);
    }


    /**
     * Check to see if the input file specified by the user exists.
     *
     * @param inputFile the name of the input file
     */
    public static void setFN(String in){
        //System.out.println("setInFileName: "+in);

        if(!isCreated){
            inF = new File(in);
            if (!inF.exists()){
                System.out.println("WPSM exception: Input File in incorrect format.  In
File: File "+in+" does not exist.\nSorry! error(s) is/are found, WPSM compiling
process aborted.");
                System.out.println ("WPSM Hint: If you don't declare the input file and
output file names first there is a chance that the default values of \"in.txt\" and
\"out.xml\" might be used.");
                System.exit(0);
            }
            else{
                in_File = in;
                }
        }
        else
            System.out.println("WPSM WARNING: User cannot change in file name after
user has created first record");
    }


    /**
     * @return the input file name
     */
    public static String getFN(){
        return in_File;
    }


    /**
     * Check to see if the output file specified by the user exists
     */
    public static void setOFN(){
         setOFN(out_File);
    }


    /**
     * Check to see if the output file specified by the user exists
     *
     * @param outFile the name of the input file
     */
    public static void setOFN(String outputFile){
        if(!isCreated){
                boolean foundfile = true;
                boolean ex = false;
            outF = new File(outputFile);
            int i = 1;
            String tempout = outputFile;
            if(outF.exists())
                foundfile = false;
            else
                out_File = outputFile;

                while(!foundfile)
                {
                        String out = "";
                        String ext = "";
                        for(int j = 0; j < outputFile.length(); j++)
```

```
                        {
                                if(outputFile.charAt(j) == '.')
                                        ex = true;
                                if(!ex)
                                        out = out + outputFile.charAt(j);
                                else
                                        ext = ext + outputFile.charAt(j);
                        }

                        ex = false;
                        out += i;
                        out += ext;
                        i++;
                    outF = new File(out);
                        System.out.println("WPSM WARNING: Out File: File "+tempout+"
already exists. Trying to use alternative " + out);
                        tempout = out;
                    if(!outF.exists())
                    {
                                foundfile = true;
                                out_File = out;
                                System.out.println("WPSM WARNING: Out File: Found
alternative " + out);
                        }
                }
        }
        else
            System.out.println("WPSM WARNING: User cannot change out file name after
user has created first record");

    }


    /**
     * @return the output file name
     */
    public static String getOFN(){
        return out_File;
    }


    /**
     * Sets the delimeter to be used for parsing the input file.
     *
     * @param delim the delimeter to use
     */
    public static void setDelimeter(String delim){
        if(!isCreated){
            delimeter = delim;
        }
        else
            System.out.println("WPSM WARNING: User cannot change delimeter after user
has created first record");
    }


    /**
     * @return the delimeter
     */
    public static String getDelimeter(){
        return delimeter;
    }


    /**
     * Sets the root name for the XML file.
     * The default value for root (RT) is "root".
     *
     * @param rootName the name for root
     */
    public static void setRT(String rootName){
        if(!isCreated2){
            RT = rootName.replaceAll(" ", "_");
        }
        else
            System.out.println("WPSM WARNING: User cannot change RT after user has
created first record");
```

```
    }


    /**
     * @return the root name
     */
    public static String getRT(){
        return RT;
    }


    /**
     * Sets the element name for the XML file.
     * The default value for element (RN) is "element".
     *
     * @param elementName the name for the element
     */
    public static void setRN(String elementName){
        if(!isCreated2)
            RN = elementName.replaceAll(" ","_");
        else
            System.out.println("WPSM WARNING: User cannot change RN after user has
created first record");
    }


    /**
     * @return the element name
     */
    public static String getRN(){
        return RN;
    }

    /**
    * Internal method called by miq.
    * Opens buffered readers and writeres.
    */
    public static void openFile(){
        if (!isCreated){
                isCreated = true;

        try {

            fr = new FileReader (in_File);
            inFile = new BufferedReader (fr);
            fw = new FileWriter (out_File);
            bw = new BufferedWriter (fw);
            outFile = new PrintWriter (bw);

        }

        catch (FileNotFoundException exception){
            System.out.println("WPSM exception: The file "+in_File+" was not
found\nSorry! error(s) is/are found, WPSM compiling process aborted.");
                System.exit(0);
        }
        catch (IOException exception){
            System.out.println("WPSM exception: " + exception +"\nSorry! error(s)
is/are found, WPSM compiling process aborted.");
        }
        }
        else
            System.out.println("WPSM WARNING: Trying to open a file twice");
    }


    /**
     * Closes the output XML file file readers and writers.
     */
    public static void closeFile(){
    if (isCreated){
        try{
            if(isCreated2)
            outFile.println("</"+RT+">");
         else
```

```
            System.out.println("WPSM WARNING: File was opened but no records were
created: "+out_File);
            inFile.close();
            outFile.close();

            System.out.println("WPSM Message: Output file has been created:
"+out_File);
        }
        catch (FileNotFoundException exception){
            System.out.println("WPSM exception: The file "+out_File+" was not
found\nSorry! error(s) is/are found, WPSM compiling process aborted.");
            System.exit(0);
        }
        catch (IOException exception){
            System.out.println("WPSM exception: " + exception +"\nSorry! error(s)
is/are found, WPSM compiling process aborted.");
        }
        isCreated = false;
    }
    else
         System.out.println("WPSM WARNING: No out file was created because no records
were created.");
    }


    /**
     * Sets the attributes for the output XML file.
     *
     * @param attri Vector containing the attributes
     */
    public static void setRA(Vector attri){
        //System.out.println("setRA: isCreated2"+isCreated2);
        String atmp = "";
        if(!isCreated2){
            //System.out.println ("Attri "+attri);
            for (int i = 0; i < attri.size(); i++){
                atmp = ""+attri.elementAt(i);
                atmp = atmp.replaceAll(" ", "_");
                attri.setElementAt(atmp,i);
         }
         RA = attri;
            //System.out.println ("RA " +RA);
        }
        else
            System.out.println("WPSM WARNING: User cannot change RA after user has
created first record.");
        }


    /**
     * Sets a particular index in the attribute vector.
     *
     * @param attri String to add
     * @param index the index to add it
     */
    public static void setRA(String attri, int index){
        if(!isCreated2){
            if(index < RA.size()){
                attri = attri.replaceAll(" ", "_");
                RA.setElementAt(attri, index);
                }
            else{
                System.out.println("WPSM exception: RA Error - RA size is
"+RA.size()+"; User trying to access RA["+index+"]\nSorry! error(s) is/are found, WPSM
compiling process aborted.");
                System.exit(0);
            }
        }
        else
            System.out.println("WPSM WARNING: User cannot change RA after user has
created first record.");
    }


    /**
     * Appends to the attribute Vector.  (User can append only to the end of
     * the attribute Vector.)
```

```
     *
     * @param attriName the name of the attribute to be appended
     */
    public static void addRA(String attriName){
        if(!isCreated2)
            RA.addElement(attriName.replaceAll(" ", "_"));
        else
            System.out.println("WPSM WARNING: User cannot change RA after user has
created first record.");
    }


    /**
     * @return the attribute vector
     */
    public static Vector getRA(){
        return RA;
    }


    /**
     * @return the attribute value
     */
    public static String getRA(int n){
        if (RA.size() > n)
            return RA.elementAt(n)+"";
        else{
                System.out.println("WPSM exception: RA Error - RA size is
"+RA.size()+"; User trying to access RA["+n+"]\nSorry! error(s) is/are found, WPSM
compiling process aborted.");
            System.exit(0);
            return null;
        }

    }


    /**
     * Sets a particular index in the record vector.
     *
     * @param value String to add
     * @param index the index to add it
     */
    public static void setRV(String value, int index){
        if (index < RV.size())
            RV.setElementAt(value, index);
        else{
                System.out.println("WPSM exception: RV Error: RV size is "+RV.size()+";
User trying to access RV["+index+"]\nSorry! error(s) is/are found, WPSM compiling
process aborted.");
            System.exit(0);
        }
    }


    /**
     * Sets the RV values.
     *
     * @param values Vector containing the record values
     */
    public static void setRV(Vector values){
        RV.removeAllElements();
        RV = values;
    }



    /**
     * Appends to the record Vector.
     *
     * @param recordValue the value of the record to be appended
     */
    public static void addRV(String recordValue){
        RV.addElement(recordValue);
    }
```

```
    /**
     * Deletes entry in the record Vector.
     *
     * @param recordValue the name of the value to be deleted
     */
    public static void removeRV(int index){
        if(index < RV.size())
            RV.removeElementAt(index);
        else {
                System.out.println("WPSM exception: RV Error: RV size is "+RV.size()+";
User trying to access RV["+index+"]\nSorry! error(s) is/are found, WPSM compiling
process aborted.");
            System.exit(0);
        }
    }


    /**
     * Deletes entry in the attribute Vector.
     *
     * @param index the index to remove
     */
    public static void removeRA(int index){
        if (!isCreated2){
            if(index < RA.size())
                RA.removeElementAt(index);
            else{
                System.out.println("WPSM exception: RA Error - RA size is
"+RA.size()+"; User trying to access RA["+index+"]\nSorry! error(s) is/are found, WPSM
compiling process aborted.");
            System.exit(0);
            }
        }
        else{
            System.out.println("WPSM WARNING: Can't make changes to RA after record
have been created");
            System.exit(0);
        }
    }


    /**
     * @return the record vector
     */
    public static Vector getRV(){
        return RV;
    }


    /**
     * @return the record value
     */
    public static String getRV(int n){
        if (RV.size() > n)
            return RV.elementAt(n)+"";
        else{
            System.out.println("WPSM exception: RV Error: RV size is "+RV.size()+";
User trying to access RV["+n+"]\nSorry! error(s) is/are found, WPSM compiling process
aborted.");
            System.exit(0);
            return null;
        }
    }


    /**
     * @return RV's length
     */
    public static int getRVSize(){
        return RV.size();
    }

    /**
     * @return RA's length
     */
    public static int getRASize(){
        return RA.size();
```

```
    }


    /**
     * Reads a line from the input file.
     *
    * !! @param lineNum the number of the line to read !!
     * @return         the string that is read
     */
    public static String readLine(){
        if (!isCreated){
            System.out.println("WPSM WARNING: ReadLine:  No Output File set, using
default");
            setFN();
         setOFN();
         openFile();
        }

        try{
            if ((line = inFile.readLine()) != null)
            line = line;
        else
            line = "EOF";
        }
        catch (FileNotFoundException exception){
                System.out.println("WPSM exception: The file "+in_File+" was not
found.\nSorry! error(s) is/are found, WPSM compiling process aborted.");
                System.exit(0);
        }
        catch (IOException exception){
            System.out.println("WPSM exception: " + exception + "\nSorry! error(s)
is/are found, WPSM compiling process aborted.");
            System.out.println (exception);
                System.exit(0);
        }

        return line;
    }


    /**
     * Checks whether a specific line is EOF.
     *
     * @return true    if line is EOF
     */
    public static boolean isEOF(String inLine){
        if (inLine == null)
                return true;

            else
                return false;
    }

    public static void createRecord(String record){
        createRecord(toVector(record));
        }

    public static void createRecord(){
    if (!isCreated){
        setFN();
        setOFN();
        openFile();
    }
    if(!isCreated2){
        outFile.println(prolog);
        outFile.println("<"+RT+">");
        isCreated2 = true;
    }

    if (RA.size() == 0){
                System.out.println("WPSM exception: Create Rocord - RA can't be
null\nSorry! error(s) is/are found, WPSM compiling process aborted.");
        System.exit(0);
        }
    else if (RV.size() == 0)
                System.out.println("WPSM WARNING: Create Rocord - Cannot create null
record; create ignored");
```

```
    else{
            if (RA.size() != RV.size())
                System.out.println("WPSM WARNING: Create Record - Attribute and record
size does not match up.");

            outFile.print("<"+RN+" ");
            for(int i = 0; i < RA.size(); i++){
                if (i < RV.size())
                    outFile.print(RA.elementAt(i)+"=\""+RV.elementAt(i)+"\" ");
                else
                    outFile.print(RA.elementAt(i)+"=\" \" ");
            }
            outFile.println("/>");
        }
    }


    public static void createRecord(Vector values){
    if (!isCreated ){
        setFN();
        setOFN();
        openFile();
    }
    if (!isCreated2){
        outFile.println(prolog);
        outFile.println("<"+RT+">");
        isCreated2 = true;
    }
    if (RA.size() == 0){
            System.out.println("WPSM exception: Create Rocord - RA can't be
null\nSorry! error(s) is/are found, WPSM compiling process aborted.");
        System.exit(0);
    }
    else if (values.size() == 0)
        System.out.println("WPSM WARNING: Create Record - Cannot create null record;
create ignored.");
    else{
            if (RA.size() != values.size())
                System.out.println("WPSM Warning: Create Record - Attribute and record
size does not match up.");

            outFile.print("<"+RN+" ");
            for(int i = 0; i < RA.size(); i++){
                if (i < values.size())
                    outFile.print(RA.elementAt(i)+"=\""+values.elementAt(i)+"\" ");
                else
                    outFile.print(RA.elementAt(i)+"=\" \" ");
            }
            outFile.println("/>");
    }
        }



    public static Vector readToVector(){
        return toVector(readLine());

    }

    /**
     * Changes a string to a Vector based on the delimeter.
     *
     * @param inputStr the string to be parsed
     * @return                the Vector of the parsed values
     */
    public static Vector toVector(String inputStr){
        Vector tmp = new Vector();
        String tmpS = "";
        String test = delimeter+""+delimeter;
        String test2 = " "+delimeter+" "+delimeter;
        String test3 = "\""+delimeter+"\"";
        String test4 = "&hello";

    inputStr = inputStr.replaceAll(test, test2);


        inputStr = inputStr.replaceAll(test3, test4);
```

```
    inputStr = inputStr.replaceAll("\"", "");   //take away the quotes that are not
around commas with ""

        tokenizer = new StringTokenizer (inputStr, delimeter);

        while (tokenizer.hasMoreTokens()){
            tmpS = tokenizer.nextToken();
            tmpS = tmpS.replaceAll(test4, ",");
            tmp.addElement(tmpS);
        }

        return tmp;
        }

    public static String toString(Vector v){
        String str = "";
        for (int i= 0; (i+1) < v.size(); i++)
            str += v.elementAt(i) + delimeter;
                str += v.elementAt(v.size() - 1);
        return str;
    }

//--------------------------------------------------------------------------
// Additions to WPSM
// not integrated in front-end yet
//--------------------------------------------------------------------------

    /**
     * Sets the prolog.
     * How to check if correct format?
     *
     * @param prologNew the new prolog value
     */
    public static void setProlog (String prologNew){
        if (!isCreated2){
            // need to check if it is in correct format
            prolog = prologNew;
        }
        else
            System.out.println("User cannot change prolog after user has created first
record");
    }


    /**
     * @return the prolog
     */
    public static String getProlog(){
            return prolog;
    }


    /**
     * Adds comments to the XML output file.
     *
     * @param comment the comments to be added to the output XML file
     */
    public static void addComment(String comment){
        if (!isCreated2){
            outFile.println("<! "+comment+" !>");
        }
        else
            System.out.println("User cannot add comments after user has created first
record");
    }


    /**
    * @return the comments
    */
    public static String getComment(){
        return comments;
    }

}
```

Appendix D.

Test Plan: Checking for dependencies

Basic Steps:

```
//only before openFile();
1 - setFN("in.txt") || setFN();
2 - setOFN ("out.txt") ||  setOFN();

//optional
3 - setDelimeter (",");

4 - openFile();

//can set before or after openFile() depending on how you set it
//if reading 1st line of file, must openFile() first
5a ?setRN();//or should we allow the user to change RN at anytime?
5b - setRA();
5c - setRT();

//optional only after openFile() and setRA()
6 - createRecord();  //can setRV first or not; setRV() can be used anytime

//must do to end program
7 - closeFile();


//helping methods only after openFile()
1 ?readLine()
2 ?readToVector()
3 ?isEOF()

//other
1 ?all get methods
2 ?toVector()
3 ?toString()


//new features not supported by front-end
//should be done before openFile()
1 ?setProlog(); getProlog();
2 ?setComments(); getComments();
```