Programming Languages & Translators

(COMS W4115)

Department of Computer Science

Columbia University

Summer 2007

XML Document Manipulation Language (XDML)

Final Report

Luba Leyzerenok
ll2310@columbia.edu
August 8, 2007

1. Abstract

XDML is a simple, high-level language for XML documents manipulation and analysis. This paper will discuss the use of the language, features that I intend to implement and its syntax. I will provide a sample XDML program in the last section.

2. Introduction

XML (the Extensible Markup Language) is a W3C-endorsed standard markup language for documents containing structured information. Its main purpose is to facilitate the sharing of data across different information systems, particularly via the Internet.

In the last few years, XML has been adopted in fields as diverse as law, aeronautics, finance, insurance, robotics, multimedia, hospitality, travel, art, construction, telecommunications, software, agriculture, physics, etc. XML has become the syntax of choice for newly designed document formats across almost all computer applications. It's used on various OS, including Linux, Windows, Mac OS, and many others.

With such wide use of XML, there is obviously a need for a language/technology that lets users manipulate XML documents. There is a wide range of technologies and applications available for programmers' use with XML. XDML is a simple, high-level language for XML document manipulation that can be used by people in difference fields, who do not have any prior experience with computer programming.

XDML will be written entirely in Java, with use of existing Java APIs, such as DOM, SAX and JAXP. The compiler will convert the XDML source code into Java code that will be compiled by a Java compiler and executed by Java virtual machine.

3. Functionality

XDML provides an easy way for users extract data from XML objects. I intend to implement the following functionality:

- Count elements: given the path to an element, such as "Class/Student" for example, the count operator will return a number of Student elements in the Class element.
- Print – this function will print the value given the path to an element.

4. Language Features

- *Simple* – The language targets audience are non-programmers from different industries. Therefore, people without programming experience should be able to read and understand the intent of the code.
- *Intuitive* – Keywords and syntax are carefully chosen to imitate natural for users (English) language. Reading the code aloud should resemble the instructions dictated by a person to another person.
- *Portability* – Since the XDML source code is translated into Java code and eventually into Java byte code and executed in Java virtual machine, XDML programs can be executed on any platform and that has Java virtual machine.

5. Syntax

5.1 Data types

The following basic data types will be available in XDML:

- *Integer* **–** will hold an integer value and identified by an **`int`** keyword.
- *String* **–** will hold one or more characters and identified by a **`string`** keyword.
- *Element* **–** will hold a reference to an XML element and identified by an **`element`**

5.2 Programming Constructs

Loops and flow control structures will be available and have similar syntax to other high-level languages, in particular, Java.

6. XDML Program Example

```
start ()
{

element class = "<Class>
                    <Student>
                        <FirstName>Jane</FirstName>
                        <LastName>Miller</LastName>
                        <SSN>987-65-4321</SSN>
                        <Phone>112-112-1212</Phone>
                    </Student>
            </Class>";


element student1 =  "<Student>
                        <FirstName>John</FirstName>
                        <LastName>Smith</LastName>
                        <SSN>123-45-6789</SSN>
                        <Phone>112-112-1212</Phone>
                        <Phone>334-334-3434</Phone>
                    </Student>";

// returns 1
int students =count (class , "Class/Student");

//prints a Student element
print (class , "Class/Student");

// prints Phone 112-112-1212 and phone 334-334-3434.
print (student1 , "Student/Phone");

}
```

7. Language Tutorial

XDML is a simple, high-level language for XML documents manipulation and analysis. XDML interpreter takes a source code written in XDML language as input.

An XDML source program begins with the word “start” and is enclosed in a pair of curly braces. XDML has 3
basic types: int, string and element. Variable of each type can be declared and assigned as follows:

```
int students = 10;
string name= “Joe”;
element student = “<Student>Joe<\Student>”
```

The print function takes 2 parameters: the variable of type element and the variable of type string. Print searches the element variable for the presence of string element and prints out all occurrences of string in the element. Here is how print would be used:

```
element class = “<Class>
                    <Student>
                        <FirstName>Jane</FirstName>
                        <LastName>Miller</LastName>
                        <SSN>987-65-4321</SSN>
                        <Phone>112-112-1212</Phone>
                    </Student>
            </Class>”;


element student1 =  “<Student>
                        <FirstName>John</FirstName>
                        <LastName>Smith</LastName>
                        <SSN>123-45-6789</SSN>
                        <Phone>112-112-1212</Phone>
                        <Phone>334-334-3434</Phone>
                    </Student>”;

//prints a Student element
print (class , “Class/Student”);

// prints Phone 112-112-1212 and phone 334-334-3434.
print (student1 , “Student/Phone”);
```

8. Language Reference Manual

A. Introduction

This manual describes XDML language.

B1. Lexical Conventions

B2. Tokens

There are five classes of tokens: identifiers, keywords, string literals, operators and other separators.

B3. Comments

The characters // introduce a comment and comment continues until the end of the line.

B4. Identifiers

An identifier is a sequence of letters, digits and an underscore. The first character must be an either letter or underscore. An identifier may have any length.

B5. Keywords

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

`element`	**`for`**	**`if`**
`else`	**`int`**	**`string`**

B6. Constants

There are several kinds of constants:

constant:

integer-constant

string-constant

element-const

B5.1 Integer constant

An integer constant consists of a sequence of digits and taken to be decimal.

B5.2 String constant

A string constant is a sequence of characters surrounded by double quotes as in "…". To represent characters such double quote, newline, backslash and some other characters inside the string constant, the following escape sequence may be used:

Newline	\n
Backslash	\\
Question mark	\?
Less than	\<
Grater than	\>

B5.3 Element constant

An element constant is a sequence of characters surrounded by "<…>".Newlines are not allowed in the element constants.

C. Basic Types

There are several fundamental types. They are integers, strings and elements. The integer type is a 32bit integer.

D. Expressions

D1. Primary Expressions

Primary expressions are identifiers, constants, strings, or expressions in parentheses.

primary-expression:

identifier

constant

string

(expression)

D2. Postfix Expressions

The operators in postfix expressions group left to right.

postfix-expression:

primary-expression

postfix-expression("string")

postfix-expression.identifier

Function call is a postfix expression, called the function designator, followed by parenthesis containing a string variable or literal.

E. Operators

E1. Unary operator " – "

An operand of the unary " – " operator must have a type **`int`**. The result is negative of the operand.

E2. Additive Operators " + " and " – "

The additive operators + and – group left-to-right. The operands of the + and – must have type **`int`**. The result of the + operator is the sum of the operands. The result of the – operator is the difference of the operands.

E3. Equality operator

The = = operator group left-to-right and evaluates to either 0 or 1. 0 is returned when the operands are equal; 1 otherwise.

E4. Assignment operator

The assignment operator groups right-to-left. The operands must be of the same type.

assignment-expression:

unary-expression assignment-operator assignment-expression

F. Declarators

Declarators are used to specify the data types of an identifier.

F1. Variable declarations

Declarations have the syntax:

Declaration

Type identifier

G. Statements

Statements are executed in sequence and fall into one of the following categories:

statement:

expression-statement

selection-statement

iteration-statement

G1. Expression statement

Expression statement have the form

expression-statement:

expression

G2. Selection statement

Selection statements choose one of several flows of control. In both forms of the if statement, the expression, which must have integer type, is evaluated and if it compares equal to 0, the first substatement is executed.

Selection-statement:

if (expression) statement

if (expression) statement else statement

G3. Iteration statement

Iterations statements specify looping.

iteration-statement:

for(expression; expression; expression;)

In the for statement, the first expression is evaluated once, and thus specifies initialization for the loop. The second expression is evaluated before each iteration and if it becomes equal 0, the loop is terminated. This expression must evaluate to an integer data type. The third expression is evaluated after each iteration and this specifies re-initialization for the loop.

H. Input and Output

XDML program can take a file name, a string, as input. Input of any other type will be discarded.

To read an input file, a file has to be open first. Open call has the following syntax:

open (filename);

Reading a file into a variable of element type, has the following syntax:

read(filename);

To print to standard output, the syntax is as follows:

print (string);

To close a file:

close (filename);

9. Project Plan

Process used for planning, specification, development and testing.

Since writing a compiler (or an interpreter, in my case) was a new task for me, in the beginning I did not have a clear idea what it entails. So aside from lectures and notes, I started with reading a guide to ANTLR: "The Definitive guide to ANTLR" by Terence Parr. In addition, I used the authors' and Prof. Edward's implementation of a small language from the appendix A of a dragon compiler book as an example and a template. Since I didn't have a clear idea of how the compiler is written, I tended to go from specification to development then to testing and back. I was trying to get the compiler working by working on little pieces and the moving on to the next step once the small part is working.

Programming Style Guide.

Since this was an individual project, I used the following style that I am the most accustomed to that is based on Sub's recommendations that can be found here: http://java.sun.com/docs/codeconv/.

Indentation:

8 spaces (2units of 4) was used as indentation.

Comments:

1. This is a block comment:

```
/*
* This is a block comment.
*/
```

2. End-of-line comment:

```
// This is an end of line comment.
```

Declarations:

One declaration per line was used.

Statements:

0. Each line contains at most one statement.
1. The enclosed statement is intended one more level then the compound statement.

2. The opening brace is at the end of the line that begins the compound statement; the close brace begins a line and indented to the beginning of the compound statement.

Name Conventions:

1. Class names are nouns that start with a capital letter.
2. Method names are verbs that start with a lower case letter and have the first letter of each internal words capitalized.
3. Names of constants consist of upper case letters.

10. Project Timeline

Brainstorming and idea generation; White paper;	June 11,2007
Revised project proposal	June 18, 2007
Language Reference Manual	June 27, 2007
Lexer	July 4, 2007
Parser	July 15, 2007
Walker	August 1, 2007
Code Generation	August 8, 2007
Completing, Testing, Final Report	August 10, 2007

11. Roles and responsibilities

This was an individual project. Therefore, one person was responsible for everything.

12. Software development environment.

1. Operating System

   This project was developed under the Windows XP platform

2. Java

   Java 1.5 was used. I used IntelliJ as a Java IDE.

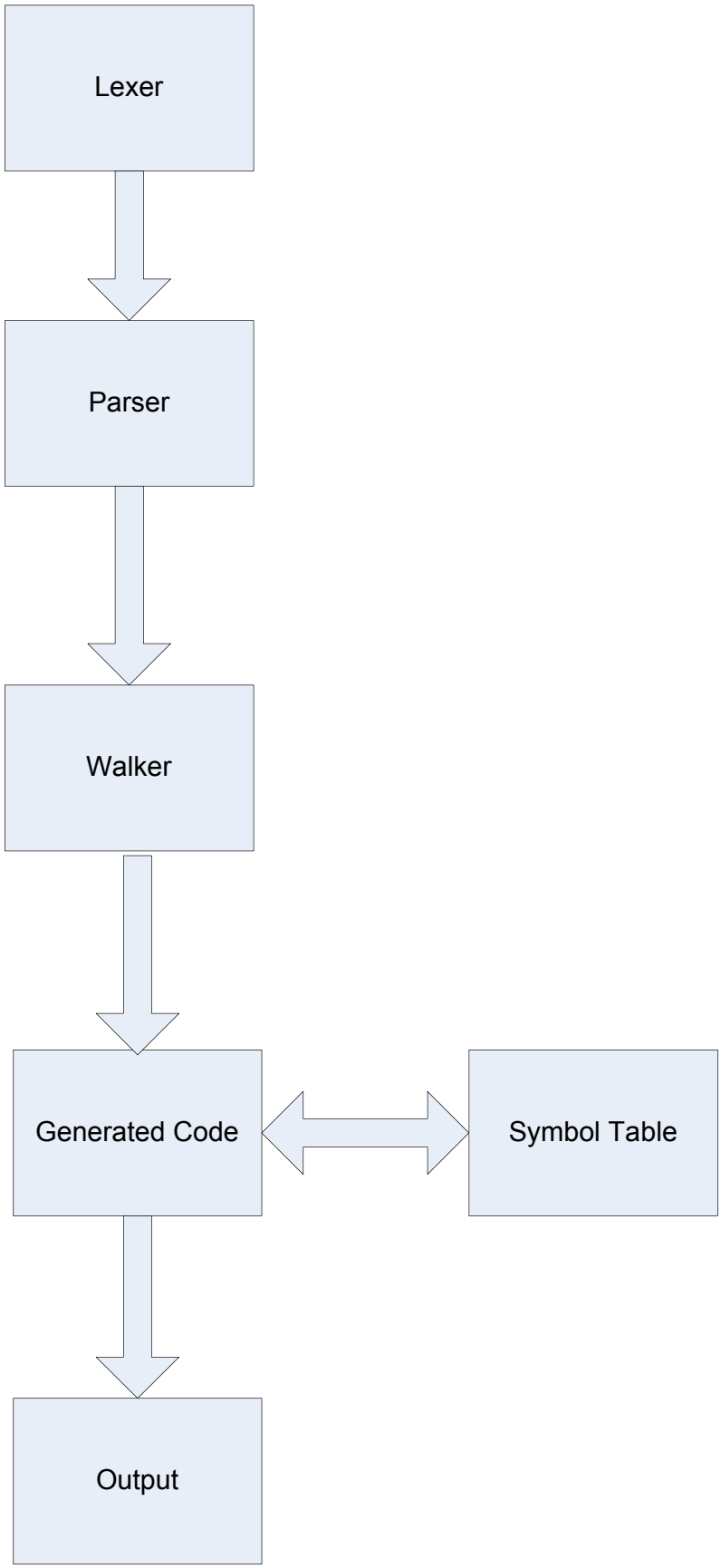
3. ANTLR

   The language lexer, parser and walker were written in ANTLR. ANTLR is a parser generator tool that can be used to implement language interpreters, compilers and other translators.

13. Project Log

Brainstorming. XDML language idea is created.	May 29, 2007
XDML language evolves. Work started on white paper.	June 5, 2007
White Paper completed.	June 11, 2007
Revising project proposal.	Jun 15, 2007
Revised white paper is completed.	June 18, 2007
Work started on Language Reference Manual.	June 21, 2007
Language Reference Manual completed.	June 27, 2007
Work on lexer started.	July 4, 2007
Work on parser started.	July 15, 2007
Modifying and Extending Lexer	July 21, 2007
Modifying and Extending Parser	July 21, 2007
Code generation started.	July 22, 2007
Work on walker started.	August 1, 2007
Lexer, Walker and Parser completed.	August 3, 2007
Code Generation	August 8, 2007
Completing, Testing, Final Report	August 10, 2007

14. Architectural Design.



15. Components Interfaces.

The XDML language consists of the following components: lexer, parser, walker, symbol table and java source code.

The XDML source program is fed into the interpreter. The lexer reads the code and translates it into a stream of tokens. The output of a Lexer is input to the Parser.

The parser takes a stream of tokens as input, analyzes the structure of the program and translates a stream of tokens into as Abstract Syntax Tree.

Walker takes an AST as input and performs various checks.

The interpreter itself reads an AST and executes the program. It looks up the variable definitions in the symbol table and executes the print() function.

Since it was an individual project, I worked on all the components.

16. Test Plan.

Example of an XDML source program:

```
start ()
{

element class = "<Class>
                    <Student>
                        <FirstName>Jane</FirstName>
                        <LastName>Miller</LastName>
                        <SSN>987-65-4321</SSN>
                        <Phone>112-112-1212</Phone>
                    </Student>
            </Class>";


element student1 =  "<Student>
                        <FirstName>John</FirstName>
                        <LastName>Smith</LastName>
                        <SSN>123-45-6789</SSN>
                        <Phone>112-112-1212</Phone>
                        <Phone>334-334-3434</Phone>
```

```
                    </Student>";

//prints a Student element
print (class , "Class/Student");

// prints Phone 112-112-1212 and phone 334-334-3434.
print (student1 , "Student/Phone");

}
```

I cannot include the target language program here because I was not able to get my interpreter fully working. Despite all my efforts, debugging and investigating, I was not unable to get my interpreter to successfully generate the AST.

Test Suit and why it was chosen

I used the above example for testing. I chose this example, because it demonstrated all main functionality of the language: the assignment and call to the print function. I did not have any automated testing.

17. Lessons Learned

- LRM is very critical in this project. The more details are thought out in LRM the easier the development will be. Design phase in general in very important and provides an opportunity to avoid many problems later in the project.
- Gained understanding of what lexer, parser and AST tree are and how they are supposed to work, even though I was not unable to get my own interpreter working.
- This was my first experience with ANTLR tool and with writing a compiler in general. Therefore, I learned concepts behind language design, what writing a compiler entails and how to code using ANTLR.

18. Advice for future students.

- I took this class during the summer semester and was doing the project individually. I think summer semester is not enough time to implement this project alone.
- Plan carefully. Good design will help to avoid many problems later in the project.
- Set realistic goals. There is a lot to learn here: language design concepts, writing grammar and parser, using ANTLR, etc.

Appendix.

XDML.g

```
class XDMLLexer extends Lexer;

options {
                testLiterals = false;
                k = 2;
                charVocabulary = '\3'..'\377';
        }

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

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

COMMENT : ( "/*" (
                options {greedy=false;} :
                (NEWLINE)
                | ~( '\n' | '\r' )
              )* "*/"
        | "//" (~( '\n' | '\r' ))* (NEWLINE)
       )                { $setType(Token.SKIP); }
      ;


NUMBER      : ('0'..'9')+
           { $setType(INT); }
           ;

STRING          : ""!
          ( ~('"' | '\n')
           | (""! "")
           )*
           ""!
           {$setType(STRING);}
           ;

ID      :       ('_'|'a'..'z'|'A'..'Z')('a'..'z'|'A'..'Z'|('0'..'9')|'_')*;


ELEMENT
   : ( '<' (' ' | '\t' )? ID (' ' | '\t' )?
         (ELEMENT)*
      "</" (' ' | '\t' )? ID (' ' | '\t' )? '>'
      )
   ;
```

```
AND   :       "&&" ;
OR    :       "||" ;
LE        :   "<=" ;
GE        :   ">=" ;
GT        :   '>' ;
LT        :   '<' ;
NE        :   "!=" ;
SEMI  :       ';' ;
LPAREN    :           '(' ;
RPAREN    :           ')' ;
LBRACE    :           '{' ;
RBRACE    :           '}' ;
LBRACK    :           '[' ;
RBRACK    :           ']' ;
ASSIGN    :           '=' ;
EQ        :       "==" ;
PLUS  :       '+' ;
MINUS     :           '-' ;
MUL   :       '*' ;
DIV       :   '/' ;
NOT       :       '!' ;
DOT       :       '.' ;
COMMA     :           ',' ;

/*****************************************************************************************/

class XDMLParser extends Parser;
options { buildAST = true; }
tokens  { NEGATE; DECLS; }

program : LBRACE^ decls (statement)* RBRACE! ;

decls : (decl)*  { #decls = #([DECLS, "DECLS"], #decls); } ;

decl : ("int" | "string" | "element") ID SEMI! ;

statement : "print"^ LPAREN! ELEMENT COMMA! STRING LPAREN!
    | SEMI
    ;

/*****************************************************************************************/

class XDMLWalker extends TreeParser;

program returns [Statement s]
{s = null;}
  : #(LBRACE
      decls
      s=statements
    )
  ;
```

```
decls
{ XDMLType t = null; }
  : #(DECLS
       (t=type ID)* )
  ;

type returns [XDMLType t]
{ t = null; }
  : ( "int"  { t = XDMLType.Int;  }
    | "string"  { t = XDMLType.String;  }
    | "element"   { t = XDMLType.Element;   }
    )
  ;

statements returns [Statement s]
{ s = null; Statement s1; }
  : s=statement (s1=statements { s = new Sequence(s, s1); } )?
  ;

statement returns [Statement s]
{ Expr e1, e2;
  s = null;
  Statement s1, s2;
}
  : #(ASSIGN e1=expr e2=expr
      { if (e1 instanceof ID) s = new Set((ID) e1, e2);
      }
    )
  | SEMI
  ;

expr returns [Expr e]
{
  Expr a, b;
  e = null;
}
  : #(PRINT  a=expr b=expr { e = new Print(a,b);})
  | INT                 { e = new Expr(#INT.getText(), XDMLType.Int); }
  | STRING                { e = new Expr(#STRING.getText(), XDMLType.String); }
  | ELEMENT                 { e = new Expr(#ELEMENT.getText(), XDMLType.Element); }

  ;
```

Expr.java

```
public class Expr extends XDMLNode {
    private String mToken;
    private XDMLType mType;

    public String getToken() {
        return mToken;
    }
```

```
    public void setToken(String pToken) {
        mToken = pToken;
    }

    public XDMLType getType() {
        return mType;
    }

    public void setMType(XDMLType mType) {
        this.mType = mType;
    }

    Expr(String pToken, XDMLType pType){
        mToken = pToken;
        mType = pType;
    }

    Expr(Expr pToken, Expr pType){
        mToken = pToken.getToken();
        mType =  pType.getType();
    }

    public void jumpTo (String test, int falseValue, int trueValue){
        if (trueValue != 0 && falseValue != 0) {
            getValue("if " + test + " goto L" + trueValue);
            getValue("goto L" + falseValue);
        } else if (trueValue != 0)
            getValue("if " + test + " goto L" + trueValue);
          else if (falseValue != 0)
            getValue("ifflase " + test + " goto L" + falseValue);

    }

    public String toString()
    {
        return mToken;
    }
}
```

ID.java

```
public class ID extends Expr {
    public ID(String pID, XDMLType pType){
        super(pID, pType);
    }
}
```

Main.java

```
import antlr.CommonAST;
import antlr.debug.misc.ASTFrame;
```

```
import java.io.*;


class Main {
   public static void main(String[] args) {
         try {

      String filename = args[0];
      if (filename == null && filename.trim().length() == 0){
         System.out.println("No argument was passed.");
      }

      FileInputStream input = new FileInputStream( filename );
            XDMLLexer lexer = new XDMLLexer(input);
            XDMLParser parser = new XDMLParser(lexer);
            parser.program();

      CommonAST mAst = (antlr.CommonAST) parser.getAST();


      System.out.println("List:\t" + mAst.toStringList());
      System.out.println("Tree:\t" + mAst.toStringTree());

      XDMLWalker walker = new XDMLWalker();
      Statement statement = walker.program(mAst);
      ASTFrame mFrame = new ASTFrame("ASTree", mAst);
            mFrame.setVisible(true);

   } catch (FileNotFoundException f) {
      System.out.println("Main.main()\tFileNotFoundException" + f.getStackTrace());
   } catch (Exception e){
      System.out.println("Main.main()\tException" + e.getStackTrace());
   }
}
```

print.java

```
public class Print extends Expr{

   public Print (Expr pType, Expr pName){
      super(pType, pName);

   }
}
```

Sequence.java

```
public class Sequence extends Statement {
   Statement mStatement1;
   Statement mStatement2;

   public Sequence(Statement pStatement1, Statement pStatement2){
```

```
      mStatement1 = pStatement1;
      mStatement2 = pStatement2;
   }

   public void gen(int b, int a){
      int label = createNewLabel();
      mStatement1.gen(b, label);
      getLabel(label);
      mStatement1.gen(b, label);
   }
}
```

set.java

```
public class Set extends Statement {
   public ID mLValue;
   public Expr mExpr;

   public Set(ID pID, Expr pExpr){
      mLValue = pID;
      mExpr = pExpr;
   }

   public XDMLType validateType (XDMLType pType1, XDMLType pType2) {
      if (pType1 == pType2)
         return pType1;
      else if (pType1.Element == pType2.Element)
         return pType1;
      else if (pType1.Int == pType2.Int)
         return pType1;
      else if (pType1.String == pType2.String)
         return pType1;
      else return null;
   }
}
```

statement.java

```
public class Statement extends XDMLNode {
   public void gen(int b, int a){}
}
```

XDMLNode.java

```
public class XDMLNode {

   void error(String pErrorMsg) {
      throw new Error(pErrorMsg);
   }
   static int labels = 0;

   public static int createNewLabel() {
```

```
        return ++labels;
    }

    public static void getLabel(int i) {
        System.out.print("L" + i + ":");
    }

    public static void getValue(String s) {
        System.out.println("\t" + s);
    }
}
```

XDMLSymbolTable.java

```
public class XDMLSymbolTable {

    private XDMLSymbolTable mParent;
    private HashMap mTable = new HashMap();

    public void XDMLSymbolTable (XDMLSymbolTable pSymTable){
        mParent = pSymTable;
    }

    public ID getID (String pName){

        for (XDMLSymbolTable tempTable = this; tempTable != null; tempTable = tempTable.mParent) {
            ID id = (ID)(tempTable.mTable.get(pName));
            if (id != null) return id;
        }
        return null;
    }

    public void addID(String pName, XDMLType pType){
        mTable.put(pName, new ID (pName, pType));
    }
}
```

XDMLType.java

```
public class XDMLType {

    public String mType;

    public XDMLType(String pType){
        mType = pType;
    }

    public static XDMLType Int        = new XDMLType("int");
    public static XDMLType String     = new XDMLType("string");
    public static XDMLType Element    = new XDMLType("element");
```

}