Programming Languages and Translators
COMS W4115
Stephen A. Edwards
Fall 2007

Orange White Green Animation Language
Final Report

Meenakshi Sripal
December 18, 2007

Table of Contents:

Introduction

Since 1996, flash technology is well known for its capability to transport graphic animations or movies to the web. Animations/movies can be delivered via Shockwave flash files, .swf, the exported Flash file format. Thousands of end-users have adopted Flash technology to generate animations. However, Adobe (formerly Macromedia) flash files can be difficult to develop due to the complexity of the design.

Therefore, I propose to develop a new programming language called Orange White Green Animation Language (OWGAL). The goal of the OWGAL programming language is to provide an easy to use Object-Oriented interface to allow for quick, comprehensive development of complex animations with additional nifty features. The generated Shockwave files can then be easily viewed by the web. OWGAL will be designed in a way that it is easy to understand with user-friendly syntax aiming users with little programming skills.

Properties of OWGAL

OWGAL language will follow the objected oriented design. Part of the animation library will be reused, such as open source codes that generate Flash animation files. Ming is planned to be used to compile the animation files turning them into .swf files. OWGAL language only allows users to load images as objects to the animation file. It will not have the drawing capability like Adobe Flash. The input images, by default, will be placed at the origin of the x-y scale at the upper left corner of the screen.

OWGAL features

-import()
-set background/foreground
-xmove/ymove
-xposition/yposition
-loop()

Program Structure

```
ROUTINE myFlash
      #width = xx
      #height = xx
      #frames = xx
{
      !!Comments
      statements
}
```

Sample Code

ROUTINE looptest
#width = 200
#height = 200
#frames = 50

```
{
	!This is comment
	!!This is also a comment

	img Meena = import (Meena.jpg, 30, 30)

	Meena x.position = 150
	Meena y.position = 150

	loop(3) {
		Meena.xmove += 5
		Meena.ymove -= 5
		Meena.xmove -= 5
		Meena.ymove +=5
	}

}
```

Language Reference Manual

1.1. Introduction

OWGAL language is a programming language that allows users to develop animations through the generation of shockwave flash files. This manual describes in details about the rules, conventions, and built-in features.

1.2. Grammar Notation

This manual introduces lexical and syntactic aspects of OWGAL's grammar. A set of productions are defined consisting of both non-terminals and terminals. The lexical grammar has undefined terminals such as if keyword, symbols like ( )[ ], else keyword; terminals are given in arial font. Examples of non-terminals are *expression* or *identifier*; non-terminals are given in *italic* font. A non-terminal are always defined on the left side of the colon whereas a mix of terminals and non-terminals are defined on the right side of the same colon. The syntactic grammar describes how sequences of tokens can form syntactically correct programs.

The input sequence of characters of the program are scanned and grouped into tokens, in which white space and comments discarded. If the sequence of characters is in the form *a*?, it denotes that the symbol *a* is optional. *a** denotes that the symbol *a* may occur zero or more times. *a*+ denotes that the symbol *a* will occur one or more times. (*a*|*b*) denotes a choice between the symbols *a* and *b*.

1.3. Lexical Conventions

In order for the OWGAL program to run, an object, mainly an image, can be imported into the code. Compiling the animation file will generate a syntactically correct program, which in turn can be used by the Java compiler. The goal of the lexical grammar is to simplify the job of the parser, which sees the animation file prior the Java compiler.

1.3.1. Tokens

There are six classes of tokens: identifiers, keywords, constants, string literals, operators, and other separators. White space, such as blanks, tabs, newlines, and comments, are ignored except as they are used to separate tokens. Some white space is required to separate otherwise adjacent identifiers, keywords, and constants.

1.3.2. Comments

There are two kinds of comments:

```
! text !
```

The character ! introduces a comment, which terminates with the character !.

```
!! text
```

Single-line comments introduced by "!!" (double exclamation point) cause the compiler to ignore the remainder of that line.

Comments do not nest, and they do not occur within string or character literals.

1.3.3. Identifiers

An *identifier* is a sequence of letters and digits. It must begin with a letter. Identifiers are case sensitive and can have any length. The underscore character (_) can be part of an identifier.

1.3.4. Keywords

The following identifiers are reserved for use and cannot be used as otherwise:

text	width	if	ROUTINE
for	length	else	return
import	img	xposition	yposition
background	foreground	set	put
while	xmove	up	down
left	right	xaxis	yaxis
ymove			

1.3.5. Constants

There are several kinds of constants.

constant: *integer-constant*
character-constant

1.3.5.1. Integer Constants

An integer constant consists of a sequence of decimal digits ranging from 0 to 9. It is unsigned and always positive. The maximum value for the integer constant is 2^{32}.

1.3.5.2. Character Constants

A character constant is a sequence of one or more characters enclosed in double quotes ("…"). Character constants do not contain the " character or newlines; in order to represent them, the following escape sequences may be used.

Newline	\n	Form feed	\f
Carriage return	\r	question mark (?)	\?
Horizontal tab	\t	double quote (")	\"
Backslash (\)	\\		

1.4. Meaning of Identifiers

Identifiers, or names, can refer to a variety of things: functions and objects. An object, sometimes called a variable, is a location in storage and its interpretation depends on its type.

1.4.1. Basic Data Types

Objects declared as text contain a sequence of characters. Objects declared as img contains special file format and attributes that are recognizable by the compiler.

1.5. Expressions

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

1.5.1. Primary Expressions

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

primary-expression: *identifier*
| *constant*
| *(assignment-expression)*

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

A constant is a primary expression, with either form of the following types: text, img, length and width.

A parenthesized expression is a primary expression containing a mix of nonterminals and terminals.

1.5.2. Additive Operators

The additive operators + and – are operated from left to right. If the operands have arithmetic type, the usual arithmetic conversions are performed.

additive-expression: *integer-expression + atom*
| *integer-expression – atom*
| *atom ;*
atom: NUMBER ;

The result of the + operator is the sum of the operators. The result of – operator is the difference of the operands.

1.5.3. Relational Operators

The relational operators are operated from left to right.

relational-expression: *relational-expression < additive-expression*
| *relational-expression > additive-expression*
| *relational-expression <= additive-expression*

| *relational-expression* >= *additive-expression*

1.5.4. Equality Operators

equality-expression: r*elational-expression*
| e*quality-expression* == r*elational-expression*
| e*quality-expression* != r*elational-expression*

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

1.5.5. Assignment Expressions

assignment-expression: p*rimary-expression*
| i*dentifier* = a*ssignment-expression*

The result for the first operand must be a variable, or a compile time error occurs. This operand must be a named variable. The type of the assignment-expression is the type of the variable. In the assignment, with =, the value of the expression replaces that of the object referred to by the i*dentifier*.

1.6. Declarations

declaration:	*TypeSpecifier InitIdentifierList ;*
type-specifier:	text
	| img
init-identifier-list:	*init-identifier*
	| *init-identifer-list, init-identifier*
init-identifer:	*identifer*
	| *identifier = string-expression ;*

A declaration consists of a *type-specifier,* followed by an *identifier,* and possibly followed by an equal sign and a s*tring-expression* or an i*nteger-expression* (if the user chooses to declare and initialize).

1.6.1. Function Declarations and Definitions

Functions are declared and defined outside of the ROUTINE block. They are type-specified by a text. Following the *type-specifier* is the *identifier*, and following the *identifier* is the parameter list surrounded by parentheses.
The syntax of the parameters is

parameter-type-list:	*parameter-list*
	| *parameter-list,…*
parameter-list:	*parameter-declaration*
	| *parameter-list, parameter-declaration*
parameter-declaration:	text *identifier*
	| img *identifier*

The function is then defined within a pair of braces. At the end of function definition, the function returns a string.

1.7. Statements

Except as indicated, statements are executed in sequence. Statements are executed for their effect, and do not have values. They fall into several groups.

statement: *expression-statement*
| *compound-statement*
| *conditional-statement*
| *iteration-statement*

1.7.1. Expression Statement

Most statements are expression statements, which have the form:

e*xpression-statement:* *(expression)* ;*

Most expression statements are assignments.

1.7.2. Compound Statement

So that several statements can be used where one is expected, the compound statement is provided:

compound-statement: { (declaration-list) (statement-list)* }*

declaration-list: *declaration*
| *declaration-list declaration*

statement-list: *statement*
| *statement statement-list*

An identifier within the declaration-list may be declared only once in the same block.

1.7.3. Conditional Statements

Conditional statements choose one of several flows of control.

conditional-statement: if (*expression*) *statement*
| if (*expression*) *statement* else *statement*

In both forms of the if statements, the expression is evaluated and the first sub-statement is executed only if the expression is not equal to zero. In the second form, the second sub-statement is executed only if the expression is zero. The else ambiguity is resolved by connecting an else with the last encountered else-less if at the same block nesting level.

1.7.4. Iteration Statements

Iteration statements specify looping:

iteration-statement: while (*expression*) *statement*

| for (*expression ; expression ; expression)*
statement
| loop (*IntExpression) statement*

In the while statement, the sub-statement is executed repeatedly so long as the value of the expression is not equal to zero; the expression must have arithmetic. With while, the test occurs before each execution of the statement.

In the for statement, the first expression is evaluated once, and thus specifies initialization for the loop. There is no restriction on its type. The second expression must have arithmetic; it is evaluated before each iteration, and if it becomes equal to zero, the for is terminated. The third expression is evaluated after each iteration, and thus specifies a re-initialization for the loop. There is no restriction on its type.

In the loop statement, the *IntExpression,* a positive integer, represents number of times to repeat the statements specified within the loop body.

1.8. OWGAL Source File Specifications

This section describes the format of an OWGAL source file.

1.8.1. ROUTINE Block Definition

Routine:	ROUTINE *identifier* *HeaderDeclaration* { *CompoundStatement* }
CompoundStatement:	*DeclarationList StatementList*
StatementList:	*Statement* | *StatementList Statement*
DeclarationList:	*Declaration* | *DeclarationList Declaration*
Declaration:	*ImageDeclaration* | *(PrimitiveTypeDeclaration)**
ImageDeclaration:	*(ImportDeclaration)+*
PrimitiveTypeDeclaration:	text *IdentifierList* | img *IdentifierList*
IdentifierList:	*Identifier* | *IdentifierList, Identifier*

The ROUTINE block definition is similar to the "main" routine declaration in a C program. The declarations, except for header declaration, and statements must be made within the ROUTINE block, and nothing should follow after the closing curly brace.

1.8.2. Header Declaration Statements

HeaderDeclaration: #width = *IntegerExpression*
#height = *IntegerExpression*
#frames = *IntegerExpression*

The header declaration allows users to specify the width and height of the animation or movie in pixels. #frames defines the number of frames for the entire animation. All must be non-negative integers. The header declaration is optional. If no header declaration is found, then the width, the height, and the frames will default to 500, 500, 100 respectively.

1.8.3. Image Declaration Statements

ImageDeclaration: img *identifier = ImportStatement*

The image declaration allows users to set up the primary objects that will be used throughout the animation. There must be at least 1 image declaration statement within the ROUTINE block in order to run. It must be declared within the ROUTINE block.

img defines a special file format that needs to be recognized by the compiler.

1.8.4. Import Statement

ImportStatement: import *(FilenameExpression, IntegerExpression, IntegerExpression)*

The import statement tells the compiler to prepare a data structure based upon another program for use by the current program and load the image onto the screen during the execution. At least one import statement must always be specified at the very beginning of the program. The 2nd parameter represents the width size of the input image. The 3rd parameter represents the height size of the input image.

The import feature provides users with ease of use and flexibility by permitting users to input the already made images, with file extensions .jpeg or .gif, into the program to work with. Images can be photos or graphics. Checks will verify that the imported image file meets the file requirements.

1.8.5. Put Declaration

PutDeclaration: put *[identifier, IntegerExpression, IntegerExpression]*

The put parameters are: object name, x coordinate, y coordinate.

The put declaration is used to place objects (images) at the specified location permanently. The object name must be taken from the identifier in the *ImageDeclaration.* The integers must be positive. There must be at least one image declaration before the put declaration can be specified. The put declaration must be declared within the ROUTINE block.

1.8.6. Set Statement

SetDeclaration: set *identifier BackFrontExpression*
BackFrontExpression: background
| foreground

The set parameters are: object name, background/foreground option.

The background option is used to place the specified object (image) behind other objects, if any. The foreground option is used to place the specified object in front of any images that are in the background layer. If set is not declared, then the images will default to the front layer and images may overlap. There must be at least one image declaration before the set statement can be specified. The set statement must be declared within the ROUTINE block.

1.8.7. Move Statement

MoveDeclaration: *identifier'.'* (xmove|ymove) *AddExpression IntegerExpression*
AddExpression: "+=" | "-="
IntegerExpression: number

The xmove and ymove keywords lets the user to dictate how the object (image) to be moved in a certain way. xmove corresponds to the x-axis or horizontal line from the graphical point of view. ymove corresponds to the y-axis or vertical line from the graphical point of view. "+=" indicates the object to be moved in the right direction. "-=" indicates the object to be moved in the left direction. Integers must be positive. There must be at least one image declaration before specifying the move statement. The move statement must be declared within the ROUTINE block.

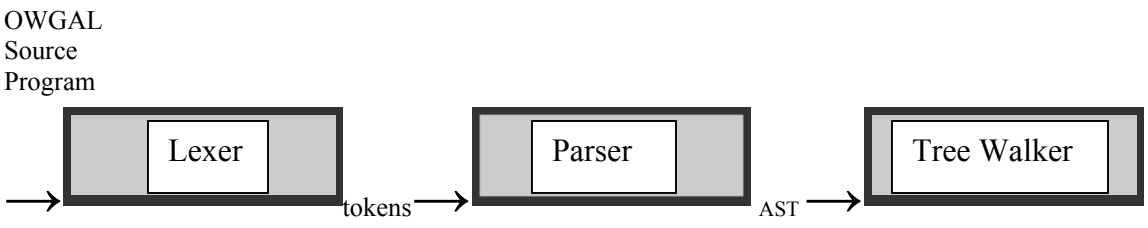
1.8.8. Position Statement

PositionDeclaration: *identifier'.'* (xposition|yposition) = *IntegerExpression*
IntegerExpression: number

The xposition and yposition keywords let the user to place the object (image) at a certain point on the screen with respect to x-axis and y-axis. xposition corresponds to the x-axis or horizontal line from the graphical point of view. yposition corresponds to the y-axis or vertical line from the graphical point of view. Integers must be positive. There must be at least one image declaration before specifying the position statement. The position statement must be declared within the ROUTINE block. If neither of them are declared, then the default position would be at 0,0.

Architectural Design

In this project, the OWGAL compiler is divided into separate parts: lexer, parser, tree walker, runtime environment, and code generator.



Front-End

The front-end is composed of lexer, parser, and tree walker. Libraries from antlr.jar are used. Lexer and parser codes are written in Antlr and reside in **grammar.g** file. The compilation of grammar.g generates OWGALLexer.java and OWGALParser.java. The tree walker resides in **walker.g** file, which generates OWGALWalker.java.

The lexer reads and scans the input source program, which has the extension .owgal. The lexer generates tokens, which are then passed to the parser to produce abstract syntax tree.

Back-End

The back-end was undetermined, unfortunately.

Test Plan

Since my project is partially-complete due to time constraint and lack of understanding and knowledge, unit testing could not be done to test the implementation of OWGAL language. However, several test cases are written below as part of the testing plan to ensure that OWGAL is a working language in terms of valid syntax and semantics. looptest program tests the loop statement to ensure that it produces the correct animation file. setTest program should fail as a result, since set statement was specified before the image declaration. imgTest program tests to ensure that it fails since no image declaration was made. The test cases should not limit to those 3, as more test cases should be written to test every logic path, and every statement in the OWGAL language.

```
ROUTINE looptest
#width = 200
#height = 200
#frames = 50
{
        !This is comment
        !!This is also a comment

        img Meena = import (Meena.jpg, 30, 30)

        Meena x.position = 150
        Meena y.position = 150

        loop(3) {
                Meena.xmove += 5
                Meena.ymove -= 5
                Meena.xmove -= 5
                Meena.ymove +=5
        }

}

ROUTINE setTest
#width = 200
#height = 500
#frames = 50
{
        !This is comment
        !!This is also a comment

        set Meena foreground;
        img Meena = import (Meena.jpg, 30, 30)
}

ROUTINE imgTest
```

```
{
	!This is comment
	!!This is also a comment

	Meena x.position = 150
	Meena y.position = 150

	loop(3) {
		Meena.xmove += 5
		Meena.ymove -= 5
		Meena.xmove -= 5
		Meena.ymove +=5
	}

}
```

Lessons Learned

Working on this project alone has been a challenging experience. Since I'm new to Antler and rusty in Java, it took tremendous amount of time for me to learn and understand the concepts behind both of these languages, especially the tree walker and java code. I should have hit the Java book at the beginning of the semester, even on the first day of classes. I feel that working with a team could have been better for me as I would be more motivated and have more resources than working alone. However, on the other hand, by working alone, it allowed me to look closely in every part (front-end and back-end) that makes up the compiler. I learned that a full, clear understanding of how the compiler works differently than the interpreter must be grasped before writing the proposal or even LRM. I should have read and understand the whole process of how graphic animations work more carefully. The design of the compiler should have taken a full consideration of the resources that are available for this project. A lesson learned here is that a better understanding of the resource requirement at the beginning would have been helpful to avoid non-realistic design. Identifying the project milestones earlier and constant visit to the milestones should be performed to stay on track. A better project documentation or log should have been kept to keep track of updates, issues, and the progress.

Appendix

```
/*
 * OWGAL.g : The main entry
 * Author   : Meenakshi Sripal
 *
 */


import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;
import antlr.RecognitionException;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;

//Invokes the ANTLR generated lexer, parser, and walker on the
//input stream

class OWGAL {
    public static void main(String[] args) {
      if(args.length != 1)
      {
        System.out.println("Usage: java OWGAL <filename>\n");
        System.exit(1);
      }

      FileInputStream input = null;

      // Get the file from input stream
      try
      {
        input = new FileInputStream(args[0]);
      }
      catch(FileNotFoundException e)
      {
        System.out.println("The specified filename '" + args[0] + "' was
not found.\n");
        System.exit(1);
      }

        try {

            // Create the lexer and parser and feed them the input
            OWGALLexer lexer = new OWGALLexer(input);
            OWGALParser parser = new OWGALParser(lexer);
            parser.startRule(); // "startRule" is the main rule in the
parser

            // Get the AST from the parser
            CommonAST parseTree = (CommonAST)parser.getAST();

            OWGALWalker w = new OWGALWalker();
            w.routine((CommonAST)parser.getAST());

            // Print the AST in a human-readable format
```

```
            //System.out.println(parseTree.toStringList());

            // Open a window in which the AST is displayed graphically
            //ASTFrame frame = new ASTFrame("AST from the OWGAL parser",
parseTree);
            //frame.setVisible(true);

        } catch(Exception e) {System.err.println("Exception: "+e);}
    }
}

/*
 * grammar.g : The lexer and parser in ANTLR grammar
 * Author   : Meenakshi Sripal
 *
 */

//Lexer & Parser
class OWGALLexer extends Lexer;

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

protected
LETTER : 'A'..'Z' | 'a'..'z' ;

protected
DIGIT : '0'..'9' ;

NUMBER : (DIGIT)+ ;

COLON : ':';
COMMA : ',';
SEMI : ';';
PERIOD : '.';
US : '_';
POUND : '#';

PLUS : '+';
PLUSEQ: "+=";
SUB : '-';
SUBEQ : "-=";
MULT : '*';
DIV : '/';
EQUAL : '=';
EQ : "==";
NE : "!=";
GT : '>';
LT : '<';
GE : ">=";
LE : "<=";
AND : "&&";
OR : "||";
```

```
LP : '(';
RP : ')';
LB : '{';
RB : '}';

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

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

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

ID options { testLiterals = true; }
   : LETTER (LETTER|DIGIT|US)*
   ;



class OWGALParser extends Parser;

options {
      k = 2;
      buildAST = true;  //Enable AST building
      exportVocab = OWGAL;
}

tokens {
      HEADER;
      BLOCK;
      STATEMENT;
      UMINUS;
}

startRule : routine EOF!
              ;

routine : "ROUTINE"^ ID
             (header_stmt)?
              code_block
            ;

header_stmt : POUND! "width" EQUAL! expr
                  POUND! "height" EQUAL! expr
                  POUND! "frames" EQUAL! expr
                  { #header_stmt = #([HEADER, "HEADER"], header_stmt); }
              ;
```

```
code_block : LB!
                ( decl | stmts )*
             RB!
             { #code_block = #([BLOCK, "BLOCK"], code_block); }
               ;

decl : "img"^ ID EQUAL! import_stmt
       ;

import_stmt : "import"^ LP! ID COMMA! expr COMMA! expr RP!
                  ;

stmts : coord_stmt
        | loop_stmt
        | set_stmt
        | if_stmt
        | break_stmt
        | continue_stmt
        ;

coord_stmt : ID PERIOD! xy_stmt
               ;

xy_stmt : ("xposition"^ | "yposition"^) EQUAL! expr
          | ("xmove"^ | "ymove"^) (PLUSEQ^ | SUBEQ^) expr
          ;

loop_stmt! : "loop"^ expr code_block
               ;

set_stmt : "set"^ ID expr
             ;

break_stmt : "break"^
           ;

continue_stmt : "continue"^
                  ;

if_stmt : "if"^ LP! expr RP!
            code_block
            ( "else"!
                  code_block
            )?
          ;

expr : and_expr ( OR^ and_expr )*
     ;

and_expr : not_expr ( AND^ not_expr )*
         ;

not_expr : ( "not"^ )? comp_expr
         ;

comp_expr : add_expr ( (EQ^ | NE^ | GT^ | LT^ | GE^ | LE^) add_expr )?
          ;
```

```
add_expr : mult_expr ( (PLUS^ | SUB^) mult_expr )*
         ;

mult_expr : unary_expr ( (MOD^ | MULT^ | DIV^) unary_expr )*
          ;

unary_expr : (SUB! NUMBER)
                  { #unary_expr = #([UMINUS,"UMINUS"], unary_expr); }
               | atom
               ;

atom : LP! expr RP!
     | NUMBER
     | ID^
     | ("background" | "foreground")
     ;


/*
 * walker.g : The AST walker
 * Author   : Meenakshi Sripal
 *
 */

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

class OWGALWalker extends TreeParser;

options{
      importVocab = OWGAL;
}

{
      java.util.Vector words;       //lookup table for identifiers
      OWGALTree tr;
      boolean prog = false;
      boolean error = false;

      //search the input string in the words vector
      boolean IDfound(String s) {
            boolean result=false;
            for (int i=0; i<words.size(); i++){
                  if (s.equals((String) words.elementAt(i))){
                        result=true;
                  }
            }
            return result;
      }

      //add reserved words to the words vector
      void init() {
            words = new java.util.Vector();
            words.addElement(w); w = "img";
```

```
            words.addElement(w); w = "import";
            words.addElement(w); w = "xmove";
            words.addElement(w); w = "ymove";
            words.addElement(w); w = "xposition";
            words.addElement(w); w = "yposition";
            words.addElement(w); w = "loop";
            words.addElement(w); w = "set";
            words.addElement(w); w = "background";
            words.addElement(w); w = "foreground";
            tr = new OWGALTree();

      }

}

routine returns [ Stmt s ]
{
      s = null;
      Stmt s1;
}
      :     #("ROUTINE" progname:ID header s=stmts
              {
                  if(!prog)
                {
                   init();
                   prog = true;
                }
                else {
                  System.out.println("ERROR! Program already defined!");
                  error = true;
                }
              }
             )
      ;

header
{
      int s1,s2,s3,t1,t2,t3;
      Type p1 = null;
      Type p2 = null;
      Type p3 = null;
}
      :     #(HEADER p1=spec s1=expr { t1 =
Integer.parseInt(s1.getText()); }
                        p2=spec s2=expr { t2 =
Integer.parseInt(s2.getText()); }
                        p3=spec s3=expr { t3 =
Integer.parseInt(s3.getText()); }
                        { tr.createPNode(t1,t2,t3); }
             )
      ;
spec returns [OwgalType t]
{
      t = null;
}
      : (    "width"    { t = OwgalType.Width; }
      |      "height"   { t = OwgalType.Height;}
```

```
      |      "frames"   { t = OwgalType.Frame; }
        )
      ;

//integer type parameters used for frames and sizes
num returns [int i]
{
      i = 0;
}
      :     j:INT { i = Integer.parseInt(j.getText()); }
      ;

stmts returns [Stmt s]
{
      int p1,p2;
      s = null;
      String s2 = null;
      Stmt p, p1, p2;
      OwgalType e;


      if (!prog)
      {
            System.out.println("Error! ROUTINE not defined first!");
            System.exit(1);
      }

}
      :     #("import" #("img" imgname:ID) impname:ID p1:NUMBER p2:NUMBER)
            {
                  //add image file to image table...
                  //load image or check if the impname is valid file
name??
                  img imgobj = getimg(#impname.getText());

                  if (imgobj == NULL) {
                        System.out.println("Error: no image file
"+#impname.getText()+" found");
                        error = true;
                  }
                  else {
                        //open the image file and load it here...
                  }

                  if (!IDfound(#imgname.getText())) //check if img
identifier already exists
                  {
                        //test the filename format...
                        //what's the default size of the screen
                        //ensure that width and length of image are <=
default screen size and > 0 otherwise fail
                        if (p1 <= 0 || p2 <=0)
                        {
                              System.out.println("Error: can not have
negative or 0 width or length value ");
                              System.out.println("for image
"+#imgname.getText());
```

```
                        error = true;
                  }
                  words.addElement(#imgname.getText());
                  tr.addObject(new Object(count, p1, p2));
                  count++;
                  }
            else
            { System.out.println("ERROR - "+#imgname.getText()+" is
already an identifier"); error = true;}
      }
|     #("xposition" xposid:ID xposnum:NUMBER)
      {
            //check if id exists
            s = xposid.getText();
            if (IDfound(#xposid.getText())) //check if img
identifier already exists
            {
                  //do the positioning of the image code here....
                  int a = Integer.parseInt(xposnum.getText());
                  if (a > 500) a = 500;
                  if (a < 0) a = 0;
                  new Xpos(s,a);
            }
            else
            { System.out.println("ERROR - "+#xposid.getText()+" is
not an identifier"); error = true;}

      }
|     #("yposition" yposid:ID yposnum:NUMBER)
      {
            //check if id exists
            s = yposid.getText();
            if (IDfound(#yposid.getText())) //check if img
identifier already exists
            {
                  //do the positioning of the image code here....
                  int a = Integer.parseInt(yposnum.getText());
                  if (a > 500) a = 500;
                  if (a < 0) a = 0;
                  new Ypos(s,a);

            }
            else
            { System.out.println("ERROR - "+#yposid.getText()+" is
not an identifier"); error = true;}

      }
|     #(PLUSEQ xypmov:ID xypmovid:ID xypnum:NUMBER)
      {
            //check if id exists
            s = xypmovid.getText();
            if (IDfound(#xpmovid.getText())) //check if img
identifier already exists
            {
                  //do the move of the image code here....
                  //ensure specified number is within the screen
limit (500,500)
```

```
                        int a = Integer.parseInt(xpnum.getText());
                        if (a > 500) a = 500;
                        if (a < 0) a = 0;
                        tr.addPLUSEQ(new Xmove(s,a));

                  }
                  else
                  { System.out.println("ERROR - "+#xposid.getText()+" is
not an identifier"); error = true;}

            }
      |     #(SUBEQ xysmov:ID xysmovid:ID xysnum:NUMBER)
            {
                  //check if id exists
                  s = xysmovid.getText();
                  if (IDfound(#xsmovid.getText())) //check if img
identifier already exists
                  {
                        //do the move of the image code here....
                        //ensure specified number is within the screen
limit (500,500)
                        int a = Integer.parseInt(xsnum.getText());
                        if (a > 500) a = 500;
                        if (a < 0) a = 0;
                        tr.addSUBEQ(new Xmove(s,a));

                  }
                  else
                  { System.out.println("ERROR - "+#xposid.getText()+" is
not an identifier"); error = true;}

            }
      |     #("loop" loopnum:NUMBER p:stmts)
            {
                  int a = Integer.parseInt(loopnum.getText());
                  new Loop(a, p)
            }
      |     #("set" bid:ID s:expr)
            {
                  s2 = bid.getText();
                  if (IDfound(#bid.getText())) //check if img identifier
already exists
                  {
                        //if previous status is set to background and
specified status is background then notify
                        //same for foreground
                        //ensure specified status is either "background"
or "foreground"
                        tr.addTrans(new Set(s2, status));
                  }
                  else
                  { System.out.println("ERROR - "+#bid.getText()+" is not
an identifier"); error = true;}

            }
      |     #("break" { p = new Break(); })
      |     #("continue" { p = new Continue(); })
```

```
      |     #("if" e=expr p=stmts
              (p1=stmts { s = new Else(e,p,p1); } ))
      ;

expr returns [ OwgalType r ]
{
      OwgalType a,b;
      r = 0;
}
      :     #(OR        a=expr b=expr { r = new Or(a, b); } )
      |     #(AND       a=expr b=expr { r = new And(a, b); } )
      |     #("not"     a=expr          { r = new Not(a); } )
      |     #(EQ        a=expr b=expr { r = new Compare("==", a, b); } )
      |     #(NE        a=expr b=expr { r = new Compare("!=", a, b); } )
      |     #(GT        a=expr b=expr { r = new Compare(">", a, b); } )
      |     #(LT        a=expr b=expr { r = new Compare("<", a, b); } )
      |     #(GE        a=expr b=expr { r = new Compare(">=", a, b); } )
      |     #(LE        a=expr b=expr { r = new Compare("<=", a, b); } )
      |     #(PLUS      a=expr b=expr { r = new Compute("+", a, b); } )
      |     #(SUB       a=expr b=expr { r = new Compute("-", a, b); } )
      |     #(MOD       a=expr b=expr { r = new Compute("%", a, b); } )
      |     #(MULT      a=expr b=expr { r = new Compute("*", a, b); } )
      |     #(DIV       a=expr b=expr { r = new Compute("/", a, b); } )
      |     #(UMINUS    a=expr        { r = new Unary("-", a); } )
      |     NUMBER                    { r = new getNum(#NUMBER.getText(),
Type.Int); }
      |     #(ID
                  { Id i = top.get(#ID.getText());
                    if (i == null)
                        System.out.println(#ID.getText() + "is not
declared.\n");
                        r = i;
                  }
             )


//creates intermediate representation
//writeToIR defined in OWGALTree class
file {int ble;}
      : ble=routine (ble=stmts)+
{
      if (!error)
      {
            tr.writeToIR();
      }
      else {
            System.out.println("Errors found - aborting");
            System.exit(1);
      }
}
;
```