Columbia University
CS4115 Programming Languages and Translators
Professor Stephen A. Edwards
Summer 2008

CATALOG: Digital Media Organization Language

Final Report

by Leonid Velikoselskiy
ID: lv223
e-mail: lvelik@gmail.com

INTRODUCTION

In our days of increasing dependency on digital information, while our old photo albums and music records are collecting dust on shelves, the great volumes of MP3s, AVIs, JPEGs and other file formats are growing on our hard drives, filling up writable media and creating a need for more removable storage and backup hardware. We think it is time to create something that will help us organize this chaos, remove a great deal of redundancy and hopefully free up some space. Unfortunately, it is not easy to build a general tool to help people catalog their data, therefore, we propose to build a whole new language for this purpose.

All files are of certain type, which is defined by their extension and format. Files also have other properties depending on their type. For example, an image file can have *Camera Model* property and a music file can have *Artist* and *Album* properties. Properties are usually used to organize the files. Music can be in a folder named after its genre and photos can be in a folder that has a name of the event that photos were taken at, but may also contain the date of this event within the folder name.

Unfortunately, there is no simple way to translate these properties into directory structures or to easily move files from folder to folder depending on their types without manual intervention. Operating systems provide some tools, but most of the time users have to repeat same steps over and over again.

We propose to write a compiler for an object-oriented programming language that helps deal with digital media files and other files in an organized manner. This language, called CATALOG, will be simple enough for all users to be able to create custom programs for themselves quickly in order to catalog their data.

CATALOG is a programming language with a wide scope of practical application. People will be able to create programs that are custom to their own file keeping habits and directory structures. This is the flexibility that file manipulation and cataloging tools cannot provide and operating systems simply lack. The language will be content oriented and very expandable, which is essential due to increasing number of digital formats and file properties that will be created in the future.

TUTORIAL

Let's write a simple HelloWorld application in Catalog:

1. Open any text editor and write the following line:

   ```
   print "Hello World!";
   ```

2. Save the file as hello.ctl (all Catalog files must have .ctl extension)
3. Run:

   ```
   java Catalog hello.ctl
   ```

The program will output “Hello World!” on console.

You can also use print function to solve some simple arithmetic. For example:

```
print 1 + 2 * 3 + 4 + (5 – 6);
```

will print the result 10.

Some results and strings can be stored in variables, following are previous programs re-written using variables:

```
$str = "Hello World";
print $str;

$a = 5 - 6;
$result = 1 + 2 * 3 + 4 + $a;
print $result;
```

Notice ; at the end of each statement. It is required to separate end of one statement from the beginning of the second one.

Catalog supports padding 0's. If you define a variable to be equal to 001 or 015, it will preserve original padding after any arithmetic operation you perform on that number. For example:

```
$index = 001;
print = $index + 1; # will print 002;
print $index + 11;  # will print 012;
```

You can also set a variable equal to a path to an existing file, that will make the variable equal to the handler to this file and allow you to view properties of that file:

```
$file = "C:\\Images\\bridge.jpg";
print $file.Type; # will return an image
```

Notice #, after the command, this indicates the start of the comments. Everything to the right of the # is ignored by the compiler.

TUTORIAL (cont.)

Once you have the file variable, you can also copy it to different directories, move it to different directories or delete it. For example:

```
cp $file "C:\\Temp";
```

will copy bridge.jpg to C:\Temp folder.

You can also do the following:

```
mv $file “C:\\Temp”,
```

which will be equivalent of these two statements:

```
cp $file "C:\\Temp"; del $file;
```

In order to operate on many files at the same time, you need `foreach` command:

```
foreach $file in "C:\\Temp" {
      print $file.Name + " was last modified on " + $file.Modified;
};
```

The above program will go through all the files in C:\Temp folder and apply whatever commands on each file as you need.

If you wish to also go through subdirectories of C:\Temp, you need to use "inside" keyword instead of "in":

```
foreach $file in "C:\\Temp" {
       cp $file "C:\\Temp"; # copy all files to the main folder
};
```

You will be asked every time a file already exists in the location you want to copy it to whether or not you want to overwrite it.

File properties can be manipulated on. For example, you can only retrieve first 2 letters of a file name:

```
$file = "C:\\Music\\song.mp3";
print $file.Name(2); # will print "so"
print $file.Name(-2); # will print "ng"
```

Dates can be retrieved by year, month or day:

```
$file = "C:\\Music\\song.mp3";
print $file.Modified("YYYY"); # will print 2008
print $file.Modified("MM"); # will print 08
```

LANGUAGE REFERENCE MANUAL

LANGUAGE

CATALOG is a shell-like language used to write programs to manipulate digital files such as images and music. It allows programmers to easily extract and modify content-specific data from files and perform file manipulations easily.

CHARACTER SET

All CATALOG programs can be written using any eight-bit ASCII characters.

COMMENTS

Comments allowed in CATALOG are only single line comments starting with # character.
On a given line, everything to the left of # is parsed and everything to the right of # before the end of the line is ignored and treated as comments. A comment can take up a whole line.

comments: '#'(~('\n'|'\r'))*;

STRINGS

Strings are any characters surrounded by double quotation marks (including double quotation marks themselves). However, if a double quotation mark is included within a string, it must be preceeded by another double quotation mark in order to be escaped (not treated as string boundary). For example, """Hello World!""" is a string with value "Hello World!", but "two" "strings" will result in a compile error. Empty string is represented as "".

String: '"'! ('"' '"'!|~('"'))* '"'!;

NUMBERS

Only non-fractional decimal numbers are allowed in CATALOG. A number can be padded with as many 0's as desired on the left, and that padding will be preserved. For example, if number is initialized to 001, then adding 1 to it will make it equal 002, adding 10 to it will make it equal 011 and adding 100 will make it equal 101. Although unconventional, this practice is very useful in CATALOG.

Number: ('0..9')+;

LANGUAGE REFERENCE MANUAL (cont)

OPERATORS

There are 5 types of operators in CATALOG

Grouping/Separating:

- **;** separates statements, every statement must have ; after it
- **{ }** loops and conditional separators (see COMMANDS and CONDITIONAL)
- **( )** grouping operator and patterns separator (see PATTERNS)

Comparison (used to compare numerical and string values and variables):

- == equals (string or number)
- **!=** not equals (string or number)
- **>** greater than (number)
- **<** less than (number)
- **>=** greater than or equals (number)
- **<=** less than or equals (number)

Logical:

- || logical "or"
- && logical "and"

Arithmetic:

- ***** multiply numbers
- / divide numbers
- **+** add numbers (or concatenate if used with strings)
- **-** subtract numbers or negate a number

Assignment operator:

- **=** used to initialize variables or set them equal to other values or other variables
  examples: `$a = 5; $b = $c; $d = $e + 3;`

VARIABLES

Any variable must start with a dollar sign ($) and be followed by a letter or an underscore character. The rest of the characters in the variable name can be digits, letters or underscore characters.

*Variable: '$' ('_'|letter)('_'|letter|digit)**

The following are all valid variable names:

```
$_, $a, $_file, $imgFile, $song2, $image_2,
```

The following variable names are not allowed:

```
2file, $3.file, $4-file
```

LANGUAGE REFERENCE MANUAL (cont)

CATALOG has only two types, numbers and strings. However, there is no need to specify type implicitly. Variables can be initialized and their values can be set like this:

```
$i = 5;
$name = "Some String";
$song = "C:\\Music\\Pop\\FamousArtist\\NewAlbum\\NewHit.mp3"
```

PATHS and FILE NAMES

Path to a directory or to a file is described in the same way as the operating system on which the program runs describes it. A path must be a string constant.

Path: String

Names of directories are separated by either slash (/) or double backslash (\\) like this:

- `"C:\\Music\\Genre\\Artist"`
- `"c:/music/genre/artist"`
- `"/usr/local/music/artist"`

Single backslash (\) as a directory separator is not allowed. For example:

```
cp "C:\Music\DuplicateSong.mp3" "C:\ToBeDeleted";
```

will result in an error.

File names are written as file name, followed by period, followed by file extension. Name and extension can be made up of any characters except for the following: `\ / : * ? " < > |`. Extensions cannot have periods in them. Everything to the right of the last period in the full file name is considered part of the file extension.

File Name: ~('\'|'/'|':'|''|'?'|'"'|'<'|'>'|'|')*
File Extension: ~('\'|'/'|':'|''|'?'|'"'|'<'|'>'|'|')*

Examples of valid file names:

- favoriteSong.mp3
- house.jpeg
- My.Favorite.City.jpg

LANGUAGE REFERENCE MANUAL (cont)

PROPERTIES

The main purpose of CATALOG language is to work with two types of files: images and music files. However, both of these types of files also share some common properties. These general properties are:

- **`Name`** — everything to the left of the last period in full file name (String)
- **`Extension`** — everything to the right of the last period in the full file name (String)
- **`Type`** – type description for a file, directory, or folder as it would be displayed in a system file browser (String)
- **`Size`** – size of the file in bytes (Number)
- **`Modified`** - date when the file was last modified in MM/DD/YYYY format (String)

For images, the most common format used in digital cameras is JPEG, therefore, programmers will be able to extract EXIF information from a file such as the following:

- **`Make —`** camera make (String)
- **`Model —`** camera model (String)
- **`Taken —`** date the picture was taken in MM/DD/YYYY format (String)
- **`Width —`** image width in pixels (Number)
- **`Height —`** image height in pixels (Number)

For music, the most common format for storing music used is MP3, therefore, programmers will be able to extract ID3 tag information from files such as the following:

- **`Artist —`** song artist (String)
- **`Album —`** album that songs belongs to (String)
- **`Title —`** song title (String)
- **`Genre —`** genre of the song (String)
- **`Year —`** year the song was recorded (Number)
- **`Track —`** song's number in the album (Number)

Example: A program to move a file to a Pop directory if its genre is pop, will look like this:

```
$file = "C:\\Music\\FamousArtist\\Song1.mp3";
if $file.Genre == "Pop" { mv $file "C:\\Music\\Pop"; }
```

LANGUAGE REFERENCE MANUAL (cont)

PATTERNS

Parts of the property of each file can be extracted using a pattern, which is put inside parentheses after the property name:

FileNameVariable.Property(Pattern)
Pattern: (String|Number)

There are two patterns. First pattern is **DATE** pattern (always a string) formatted like this:

- YYYY or YY represent year digits. For YYYY, first Y is the millenium, second Y is the century, third Y is the decade and fourth Y is the year. For YY, first Y is the decade and second Y is the year.
- MM represents month, where 01 is January, 02 is February, 03 is March, 04 is April, 05 is May, 06 is June, 07 is July, 08 is August, 09 is September, 10 is October, 11 is September and 12 is December.
- DD represents day, any number from 01 to 31 (01 to 09 are padded with one 0 on the left)
  Some allowed Date pattern strings are: "YY", "YYMM", "DDMMYYYY", "MMDD"

  For example, the code to print year from the date picture taken property would look like this:

```
$file = "C:/oldPhotos/picFromLastYear.jpg";
print $file.Taken("YYYY"); # output would be 2007 or
print $file.Taken("YY"); # output would be 07
```

Second patter is **SUBSTRING** pattern (always a number):

- positive number represents number of characters to return from the beginning of the file name
- negative number represents the amount of characters to return from the end of the file name

  The code to print the first 4 and the last 4 characters of the file name would look like this:

```
$file = "C:/oldPhotos/picFromLastYear.jpg";
print $file.Name(4);     # output would be picF
print $file.Name(-4);    # output would be Year
```

LANGUAGE REFERENCE MANUAL (cont)

COMMANDS

- **`cp PATH PATH`** – copies file(s) from one path to another. Example: `cp "K://Music//*" "C://Music//FromUSBDrive";` will copy all files from Music folder on K drive to FromUSBDrive folder in Music folder on C drive.
- **`mv PATH PATH`** – copies file(s) from one path to another, optionally with new name(s), and deletes them from the original location. Example:
  `mv "K://Music//*" "C://Music//FromUSBDrive";`
  will copy all files from Music folder on K drive to FromUSBDrive folder in Music folder on C drive and delete all files in Music folder on K drive.
- `del PATH` – deletes file(s) from a certain location. Example:
  `cp "K://Music//*" "C://Music//FromUSBDrive";`
  `del "K://Music//*";` is equivalent to:
  `mv "K://Music//*" "C://Music//FromUSBDrive";`
- **`foreach VARIABLE (in|inside) { #do something };`**
  There are two ways to traverse through a file collection:
  - non-recursively, meaning only the files in the current directory are read (this can be done using `foreach ... in` keyword pair)
  - recursively, meaning the files in the current directory are read along with the files in all subdirectories (this can be done using `foreach ... inside` keyword pair)

  When used together with sort, loop iterates through a sorted collection of files.
- **`print (String|Number|Statement)+`** - prints any string or number to the console. Example:
  `print "Hello World!"; # output would be Hello World!`

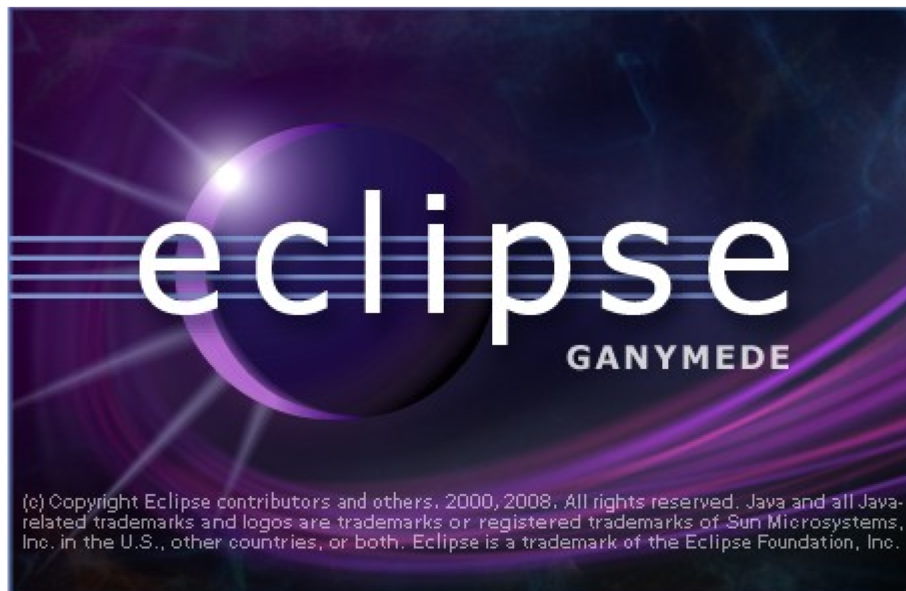
CONDITIONAL

if checks that some condition is true (some variables or constants are equal or not equal to or greater or less than each other) and then executes the code within curly brackets. In case condition is false, everything within curly brackets is not executed. If can also check for success of an operation such as assignment of a value to a variable.

PROJECT TOOLS

For Lexer and Parser:

For Java:



with

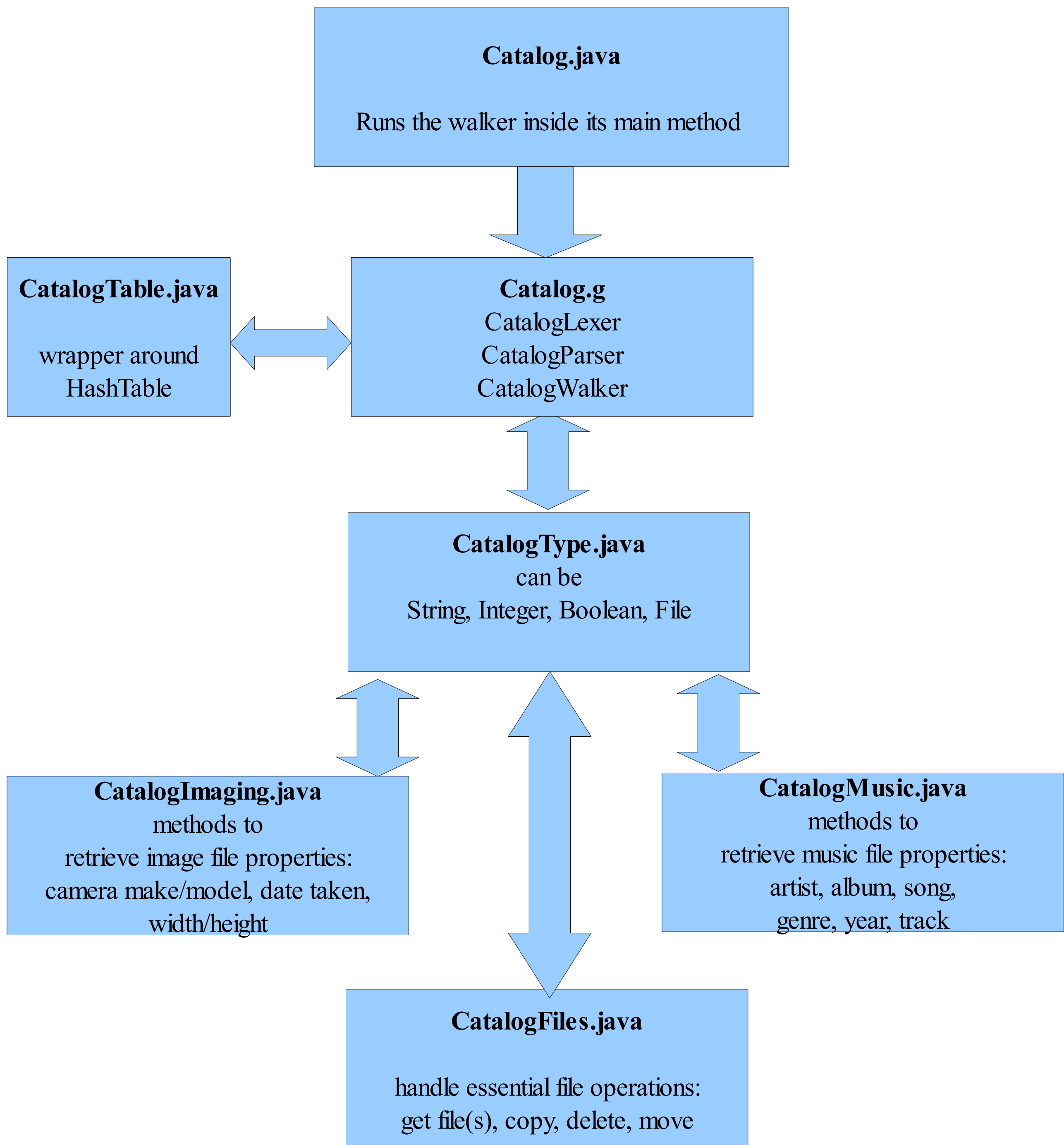
http://antlreclipse.sourceforge.net/

external libraries:

Java ID3 Tag Library

and

ARCHITECTURE



FILES (Catalog.g) - Lexer

```
class CatalogLexer extends Lexer;
options {
    k = 2;
    testLiterals = false;
    charVocabulary = '\3'..'\377';
}

// arithmetic
PLUS:   '+';
MINUS:  '-';
TIMES:  '*';
DIV:    '/';
ASSIGN: '='; // assignment operator

// grouping/separating
SEMI:   ';';
LPAREN: '(';
RPAREN: ')';
LBRACE: '{';
RBRACE: '}';
PERIOD: '.';

// comparison
EQ:     "==";
NOTEQ:  "!=";
GREAT:  '>';
LESS:   '<';
GREATEQ:">=";
LESSEQ: "<=";

// logical
AND:    "&&";
OR:     "||";
TRUE:   "true";
FALSE:  "false";

// keywords/commands
PRINT:   "print";
IF:      "if";
COPY:    "cp";
MOVE:    "mv";
DELETE:  "del";
FOREACH: "foreach";

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

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

VAR: '$' ID;
NUMBER: (DIGIT)+;
STRING: '"'! ( '"' '"'! | ~('"'))*  '"'!; // string, escape " with ")
WS: (' '|'\t'|'\n' { newline(); } | '\r') { $setType(Token.SKIP); }; // white space

COMMENTS: '#'(~('\n'|'\r'))* { $setType(Token.SKIP); }; // single line comments
```

FILES (Catalog.g) - Parser

```
class CatalogParser extends Parser;
options {
    buildAST = true;
    k = 2;
}

tokens { STATEMENTS; REVERSE; FOR; }

file: (stmt SEMI!)* EOF! { #file = #([STATEMENTS], file); };

stmt:
      PRINT^ stmt |
      IF^ or LBRACE! (stmt SEMI!)* RBRACE! | // if { ... }
      COPY^ stmt STRING | // command cp (copy files from one path to another)
      MOVE^ stmt STRING | // command mv (move files from one path to another)
      DELETE^ stmt |    // command del (delete files from a path)
      foreach | // loop
      VAR ASSIGN^ stmt | // assignment
      or;
or: and (OR^ and)*; // ||
and: eqneq (AND^ eqneq)*; // &&
eqneq: comp ((EQ^ | NOTEQ^) comp)*; // == and !=
comp: addsub ((LESS^ | LESSEQ^ | GREAT^ | GREATEQ^) addsub)*; // <, <=, >, >=
addsub: multdiv ((PLUS^ | MINUS^) multdiv)*; // + and -
multdiv: pattern ((TIMES^ | DIV^) pattern)*; // * and /
pattern: property (LPAREN^ property RPAREN!)*;
property: basic (PERIOD^ basic)*;
basic:      ID | VAR | NUMBER | STRING | MINUS^ basic { #basic.setType(REVERSE); }
|
            TRUE | FALSE | LPAREN! stmt RPAREN!; // basic element

foreach: FOREACH^ v:VAR ("in" | "inside") filelist (sort)? LBRACE! (stmt SEMI!)*
RBRACE! {
      #foreach = #([FOR, "FOR"], #foreach); };
sort: "sortby"^ ID ((PLUS|MINUS)!)?;

filelist: STRING (PERIOD^ ID)*;
```

FILES (Catalog.g) - Walker

```
class CatalogWalker extends TreeParser; {
      CatalogTable ctlTable = new CatalogTable(); // stores variables
      CatalogFiles ctlFiles = new CatalogFiles(); // essential file operations
}

file { CatalogType a; }: #(STATEMENTS (a=stmt)* );
stmts returns [CatalogType s1] { s1 = null; CatalogType s2; }: s1=stmt (s2=stmts)?;
stmt returns [CatalogType r] {
      r = null; CatalogType a = null; CatalogType b = null;
}:
#(PRINT a=stmt { System.out.println(a); r = new CatalogType(true); } ) |
#(IF a=ifCond:stmt { if (a.isTrue()) {
      r = stmts(ifCond.getNextSibling()); } } ) |
#(COPY a=stmt b=stmt { r = new CatalogType(ctlFiles.copyFile(a, b)); } ) |
#(MOVE a=stmt b=stmt { r = new CatalogType(ctlFiles.moveFile(a, b)); } ) |
#(DELETE a=stmt { r = new CatalogType(ctlFiles.deleteFile(a)); } ) |
#(FOR FOREACH  {
      AST var = #FOREACH.getFirstChild();
      AST flag = var.getNextSibling();
      String path = flag.getNextSibling().getText();
      java.util.List list =
            ctlFiles.getFiles(new CatalogType(path), flag.getText());
      for (int i = 0; i < list.size(); i++) {
            ctlTable.put(var.getText(), (CatalogType)list.get(i));
            r = stmts(flag.getNextSibling());
      }
} ) |
#(ASSIGN VAR a=stmt { ctlTable.put(#VAR.getText(), a); } ) |
#(OR a=firstOr:stmt {
      r = new CatalogType(a.isTrue() ? true :
            stmt(firstOr.getNextSibling()).isTrue()); } ) |
#(AND a=stmt b=stmt { r = new CatalogType(a.isTrue() && b.isTrue()); } ) |
#(EQ a=stmt b=stmt { r = a.equals(b); } ) |
#(NOTEQ a=stmt b=stmt { r = a.notEquals(b); } ) |
#(LESS a=stmt b=stmt { r = a.lessThan(b); } ) |
#(LESSEQ a=stmt b=stmt { r = a.lessThanOrEquals(b); } ) |
#(GREAT a=stmt b=stmt { r = a.greaterThan(b); } ) |
#(GREATEQ a=stmt b=stmt { r = a.greaterThanOrEquals(b); } ) |
#(PLUS a=stmt b=stmt { r = a.add(b); } ) |
#(MINUS a=stmt b=stmt { r = a.subtract(b); } ) |
#(TIMES a=stmt b=stmt { r = a.multiply(b); } ) |
#(DIV a=stmt b=stmt { r = a.divide(b); } ) |
#(PERIOD a=property:stmt {
      r = a.getProperty(property.getNextSibling().getText()); } ) |
#(LPAREN a=pattern:stmt {
      r = a.applyPattern(stmt(pattern.getNextSibling()).getStrValue()); } ) |
#(VAR {
      if ( !(ctlTable.containsKey(#VAR.getText()))) {
            System.err.println("unknown variable " + #VAR.getText()); }
      r = (CatalogType)ctlTable.get(#VAR.getText());
} ) |
#(NUMBER { r = new CatalogType(#NUMBER.getText()); } ) |
#(STRING { r = new CatalogType(#STRING.getText()); } ) |
#(REVERSE a=stmt { r = new CatalogType(-a.getIntValue()); } ) |
#(TRUE { r = new CatalogType(true); } ) |
#(FALSE { r = new CatalogType(false); } );
```

FILES (Catalog.java)

```
import java.io.*;

import antlr.*;
import antlr.debug.misc.*;

public class Catalog {

   private static boolean _DEBUG = false;

   public static void main(String[] args) {

      String filename = "";
      if (args.length < 1 || !(filename = args[0]).endsWith("ctl"))
      {
            System.err.println("Usage: java Catalog [your program file name].ctl");
      }

      try
      {
            FileInputStream file = new FileInputStream(filename);
            DataInputStream input = new DataInputStream(file);

            CatalogLexer lexer = new CatalogLexer(input);
            CatalogParser parser = new CatalogParser(lexer);
            parser.file();

            CommonAST ASTTree = (CommonAST)parser.getAST();
            if (_DEBUG) {
                  System.out.println(ASTTree.toStringList());
                  ASTFrame frame = new ASTFrame("AST Tree", ASTTree);
                  frame.setVisible(true);
            }
            CatalogWalker walker = new CatalogWalker();
            walker.file(ASTTree); // execute the program
      }
      catch (FileNotFoundException e) {
            System.err.println(String.format("File %s was not found", filename));
                  if (_DEBUG) { e.printStackTrace(); }
            }
            catch (TokenStreamException e) {
                  System.err.println(e);
                  if (_DEBUG) { e.printStackTrace(); }
            }
            catch (RecognitionException e) {
                  System.err.println(e);
                  if (_DEBUG) { e.printStackTrace(); }
            }
      }
}
```

FILES (CatalogTable.java)

```
import java.util.*;
public class CatalogTable extends Hashtable<String, CatalogType> {
      // this class is a type of wrapper, it has no implementation on purpose
}
```

FILES (CatalogType.java)

```java
public class CatalogType implements Comparable {

	private Object _value  = null;

	// CONSTRUCTORS
	public CatalogType() { }
	public CatalogType(Object value) { _value = value; }
		public CatalogType(String value) {
		if (CatalogFiles.fileExistsReadableAndNotDir(value)) {
			// change of plans, value is going to be a file
			_value = CatalogFiles.getFile(value);
		}
		else {
			_value = value; // otherwise, it is just a string
		}
	}
	public CatalogType(int value) { _value = new Integer(value); }
	public CatalogType(boolean value) { _value = new Boolean(value); }

	// TYPE CHECKERS

	public boolean isInt() { return this.getValue() instanceof Integer; }
	public boolean isStr() { return this.getValue() instanceof String; }
	public boolean isBool() { return this.getValue() instanceof Boolean; }
	public boolean isFile() { return this.getValue() instanceof File; }

	// VALUE GETTERS

	public Object getValue() {
		return this._value instanceof CatalogType ?
			((CatalogType)this._value).getValue() : this._value;
	}

	public int getIntValue() {
		return this.isInt() ? ((Integer)_value).intValue() : isNumber(this) ?
			 Integer.parseInt(getStrValue()) : 0;
	}

	public String getStrValue() {
		return this.getValue() != null ? this.getValue().toString() : "0";
	}

	public Boolean getBoolValue() { return (Boolean)this.getValue(); }

	public boolean isTrue() {
		return this.isBool() && (Boolean)this.getValue() == true;
	}

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

FILES (CatalogType.java) - cont

```java
public CatalogType getProperty(String propertyName) {
if (this.isFile()) {
      File file = (File)this.getValue();
      // general properties
      if (propertyName.equals("Name")) {
            return new CatalogType(CatalogFiles.getFileNameNoExtension(file));
      }
      else if (propertyName.equals("Extension")) {
            return new CatalogType(CatalogFiles.getFileExtension(file));
      }
      else if (propertyName.equals("Type")) {
            return new CatalogType(CatalogFiles.getFileType(file));
      }
      /* ... MORE ... */
      // image file properties
      else if (propertyName.equals("Make")) {
            return new CatalogType(CatalogFiles.isImage(file) ?
                   CatalogImaging.getCameraMake(file) : "not an image");
      }
      else if (propertyName.equals("Model")) {
            return new CatalogType(CatalogFiles.isImage(file) ?
                   CatalogImaging.getCameraModel(file) : "not an image");
      }
      else if (propertyName.equals("Taken")) {
            return new CatalogType(CatalogFiles.isImage(file) ?
                   CatalogImaging.getDateTaken(file) : "not an image");
      }
      /* ... MORE ... */
      // music file properties
      else if (propertyName.equals("Artist")) {
            return new CatalogType(CatalogFiles.isMusic(file) ?
                  CatalogMusic.getArtist(file) : "not a music file");
      }
      else if (propertyName.equals("Album")) {
            return new CatalogType(CatalogFiles.isMusic(file) ?
                  CatalogMusic.getAlbum(file) : "not a music file");
      }
      else if (propertyName.equals("Title")) {
            return new CatalogType(CatalogFiles.isMusic(file) ?
                   CatalogMusic.getTitle(file) : "not a music file");
      }
      else if (propertyName.equals("Genre")) {
            return new CatalogType(CatalogFiles.isMusic(file) ?
                  CatalogMusic.getGenre(file) : "not a music file");
                  }
      /* ... MORE ... */
                  return new CatalogType("unknown property");
      }
      else { return new CatalogType("not a file"); }
      }
```

FILES (CatalogType.java) - cont

```
public CatalogType applyPattern(String pattern) {

      String result = "";
      if (isNumber(pattern)) { // SUBSTRING pattern
            int substr = Integer.parseInt(pattern);
            if (substr > 0) {
                  result = this.getStrValue().substring(0, substr);
            }
            else {
               result =
                this.getStrValue().substring(this.getStrValue().length() + substr);
            }
      }
      else { // DATE pattern
            SimpleDateFormat formatter = new SimpleDateFormat("MM/dd/yyyy");
            try {
                  Date date = formatter.parse(this.getStrValue());
                  formatter = new SimpleDateFormat(
                        pattern.replace("D", "d").replace("Y", "y"));
                        result = formatter.format(date);
                  }
                  catch (ParseException e) {
                        e.printStackTrace();
                  }
            }
            return new CatalogType(result);
      }
}

// COMPARISON FUNCTIONS

public CatalogType lessThan(CatalogType obj) {
      return new CatalogType(this.isInt() && obj.isInt() &&
            this.getIntValue() < obj.getIntValue());
      }

public CatalogType lessThanOrEquals(CatalogType obj) {
      return new CatalogType(this.lessThan(obj).getBoolValue() ||
             this.equals(obj).getBoolValue());
}

public CatalogType greaterThan(CatalogType obj) {
            return new CatalogType(this.isInt() && obj.isInt() &&
                   this.getIntValue() > obj.getIntValue());
}

public CatalogType greaterThanOrEquals(CatalogType obj) {
      return new CatalogType(this.greaterThan(obj).getBoolValue() ||
             this.equals(obj).getBoolValue());
}

public CatalogType equals(CatalogType obj) {
      return new CatalogType(this.getValue().equals(obj.getValue()));
}

/* ... MORE ... */
```

FILES (CatalogType.java) - cont

```java
// ARITHMETIC FUNCTIONS

public CatalogType add(CatalogType obj) {

	if (isNumber(this) && isNumber(obj)) { // add two numbers
		return preservePadding(this.getIntValue() + obj.getIntValue(), obj);
	}
	else if (this.isInt() && obj.isStr()) { // Java concatenates strings and ints
		return new CatalogType(this.getIntValue() + obj.getStrValue());
	}
	else if (this.isStr() && obj.isInt()) { // Java concatenates strings and ints
		return new CatalogType(this.getStrValue() + obj.getIntValue());
	}
	else if (this.isStr() && obj.isStr()) { // Java concatenates strings
		return new CatalogType(this.getStrValue() + obj.getStrValue());
	}
	return new CatalogType(null);
}

public CatalogType subtract(CatalogType obj) {
	if (isNumber(this) && isNumber(obj)) { // subtract two numbers
		return preservePadding(this.getIntValue() - obj.getIntValue(), obj);
	}
	return new CatalogType(null);
}

public CatalogType multiply(CatalogType obj) {
	if (isNumber(this) && isNumber(obj)) { // multiply two numbers
		return preservePadding(this.getIntValue() * obj.getIntValue(), obj);
	}
	return new CatalogType(null);
}

public CatalogType divide(CatalogType obj) {
	if (isNumber(this) && isNumber(obj)) { // divide two numbers
		return preservePadding(this.getIntValue() / obj.getIntValue(), obj);
	}
	return new CatalogType(null);
}
```

FILES (CatalogType.java) - cont

```
// HELPER FUNCTIONS

public static boolean isNumber(String str) {
      try { // check if the string parses to integer successfully
            Integer.parseInt(str);
      }
      catch (NumberFormatException nfe){
            return false; // failed, not a number
      }
      return true; // success, string is a number
}

public static boolean isNumber(CatalogType obj) {
      return obj.isInt() || isNumber(obj.getStrValue());
}

public static String padWithZeroes(int number, int width) {
      StringBuffer result = new StringBuffer(""); // add zeroes to the left
      for (int j = 0; j < width-Integer.toString(number).length(); j++) {
            // add as many zeroes as needed to make all have same number of digits
            result.append("0");
      }
      result.append(Integer.toString(number));
      return result.toString();
}

private CatalogType preservePadding(int number, CatalogType obj) {
      if (number >= 0 && (this.getStrValue().startsWith("0") ||
                  obj.getStrValue().startsWith("0"))) {
            return new CatalogType(padWithZeroes(number,
             Math.max(this.getStrValue().length(), obj.getStrValue().length())));
      }
      return new CatalogType(number);
}

public int compareTo(Object o) {
      return
      this.getProperty("Name").compareTo(((CatalogType)o).getProperty("Name"));
}
```

FILES (CatalogImaging.java)

```
public class CatalogImaging {

public static String getCameraMake(File file) {
      Directory exifDirectory = getMetaDirectory(file);
      return exifDirectory != null ?
       exifDirectory.getString(ExifDirectory.TAG_MAKE) : "no camera make info";
}

public static String getCameraModel(File file) {
      Directory exifDirectory = getMetaDirectory(file);
      return exifDirectory != null ?
       exifDirectory.getString(ExifDirectory.TAG_MODEL) : "no camera model info";
}

public static String getDateTaken(File file) {
      Directory exifDirectory = getMetaDirectory(file);
      try {
        SimpleDateFormat formatter = new SimpleDateFormat("yyyy:MM:dd hh:mm:ss");
        if (exifDirectory != null) {
            String dateStr =
                   exifDirectory.getString(ExifDirectory.TAG_DATETIME_ORIGINAL);
            Date date = formatter.parse(dateStr);
            formatter =  new SimpleDateFormat("MM/dd/yyyy"); // desired format
            return formatter.format(date);
        }
        return "no date picture taken info";
      }
      catch (ParseException e) {
            e.printStackTrace();
      }
return "error occurred while getting the date";
}

public static String getWidth(File file) {
      Directory exifDirectory = getMetaDirectory(file);
      return exifDirectory != null ?
      exifDirectory.getString(ExifDirectory.TAG_EXIF_IMAGE_WIDTH):"no width info";
}

public static String getHeight(File file) {
      Directory exifDirectory = getMetaDirectory(file);
      return exifDirectory != null ?
      exifDirectory.getString(ExifDirectory.TAG_EXIF_IMAGE_HEIGHT):"no width info";
}

private static Directory getMetaDirectory(File file) {
      try {
            Metadata metadata = JpegMetadataReader.readMetadata(file);
            return metadata.getDirectory(ExifDirectory.class);
      }
      catch (JpegProcessingException e) {
            e.printStackTrace();
      }
      return null;
}
}
```

FILES (CatalogMusic.java)

```
public class CatalogMusic {

public static String getArtist(File file) {
      MP3File mp3file = getMP3File(file);
      return mp3file != null ? mp3file.getID3v1Tag().getArtist() :"no Artist info";
}

public static String getAlbum(File file) {
    MP3File mp3file = getMP3File(file);
    return mp3file != null ? mp3file.getID3v2Tag().getAlbumTitle():"no Album info";
}

public static String getTitle(File file) {
     MP3File mp3file = getMP3File(file);
     return mp3file != null ? mp3file.getID3v2Tag().getSongTitle():"no Title info";
}

public static String getGenre(File file) {
      MP3File mp3file = getMP3File(file);
      return mp3file != null ?
             _genreMap.get(mp3file.getID3v1Tag().getSongGenre()) : "no Genre info";
}

public static String getYear(File file) {
   MP3File mp3file = getMP3File(file);
   return mp3file != null ? mp3file.getID3v2Tag().getYearReleased():"no Year info";
}

public static String getTrack(File file) {
      MP3File mp3file = getMP3File(file);
      return mp3file != null ?
            mp3file.getID3v2Tag().getTrackNumberOnAlbum() : "no Track info";
}

public static MP3File getMP3File(File file) {
      try {
            return new MP3File(file);
      }
      catch (IOException e) {
            e.printStackTrace();
      }
      catch (TagException e) {
            e.printStackTrace();
      }
      return null;
}

private static HashMap<String, String> _genreMap;

static {
      _genreMap = new HashMap<String, String>();
      _genreMap.put("0", "Blues");
      _genreMap.put("1", "Classic Rock");
      _genreMap.put("2", "Country");
      /* ... LOTS MORE ... */
} }
```

FILES (CatalogFiles.java)

```
public class CatalogFiles {

// HELPER FUNCTIONS

public static boolean fileExistsReadableAndNotDir(String path) {
	File file = getFile(path);
	return file.exists() && file.canRead() && file.isFile();
}

public static boolean dirExistsReadableAndNotFile(String path) {
	File file = getFile(path);
	return file.exists() && file.canRead() && file.isDirectory();
}

public static File getFile(String path) {
	return new File(path);
}

public static String getFileNameNoExtension(File file) {
	int period = file.getName().lastIndexOf(".");
	return period > 0 ? file.getName().substring(0, period) : file.getName();
}

public static String getFileExtension(File file) {
	int period = file.getName().lastIndexOf(".");
	return period > 0 ?
		file.getName().substring(file.getName().lastIndexOf(".")+1) : "";
}

public static String getFileType(File file) {
	return getFileExtension(file).equals("mp3") ? "Music" :
		getFileExtension(file).equals("jpg") ||
		getFileExtension(file).equals("jpeg") ? "Image" : "Unknown";
}

public static boolean isImage(File file) {
	return getFileType(file).equals("Image");
}

public static boolean isMusic(File file) {
	return getFileType(file).equals("Music");
}

public static long getFileSize(File file) {
	return file.length();
}

public static String getFileModified(File file) {
	Date date = new Date(file.lastModified());
	SimpleDateFormat formatter = new SimpleDateFormat("MM/dd/yyyy");
	return formatter.format(date);
}
```

FILES (CatalogFiles.java) - cont

```
// MAIN FILE MANIPULATION FUNCTIONS

public boolean copyFile(CatalogType fromPath, CatalogType toPath) {

      File fromFile = getFile(fromPath.getStrValue());
      File toFile = getFile(toPath.getStrValue());

      // check if file to copy exists, if it's a file and if can be read
      if (!fileExistsReadableAndNotDir(fromPath.getStrValue())) {
            System.err.println(fromPath +
                  " does not exists or not readable or a directory.");
            return false;
      }

      if (toFile.isDirectory()) {
            toFile = new File(toFile, fromFile.getName());
      }

      if (toFile.exists()) {
         // handling of file that already exists (asking user to overwrite it)
         // ..................
      }

      // actual copying of the file
      FileInputStream from = null;
      FileOutputStream to = null;
      try {
            from = new FileInputStream(fromFile);
            to = new FileOutputStream(toFile);
            byte[] buffer = new byte[4096];
            int bytesRead;
            while ((bytesRead = from.read(buffer)) != -1) {
                  to.write(buffer, 0, bytesRead); // write
            }
      }
      catch (FileNotFoundException e) { e.printStackTrace(); return false; }
      catch (IOException e) { e.printStackTrace(); return false; }
      finally {
            if (from != null) {
                  try {
                        from.close();
                  }
                  catch (IOException e) { e.printStackTrace(); return false; }
            }
            if (to != null) {
                    try {
                      to.close();
                    }
                  catch (IOException e) { e.printStackTrace(); return false; }
            }
        }
        return true;

      }
```

FILES (CatalogFiles.java) - cont

```
public boolean deleteFile(CatalogType delPath) {

    File delFile = new File(delPath.getStrValue());

    // make sure the file or directory exists
    if (!delFile.exists()) {
      System.err.println(delPath + " does not exist");
      return false;
    }

    // make sure we have permissions to write to a file
    if (!delFile.canWrite()) {
      System.err.println("cannot write to destination file " + delPath);
      return false;
    }

    // if file is a directory, make sure it is empty
    if (delFile.isDirectory()) {
      String[] files = delFile.list();
      if (files.length > 0) {
            System.err.println("directory is not empty " + delPath);
            return false;
       }
    }

    return delFile.delete();

}

public List<CatalogType> getFiles(CatalogType dirPath, String flag) {
   if (dirExistsReadableAndNotFile(dirPath.getStrValue())) {
      File dirFile = new File(dirPath.getStrValue());
      File[] files = dirFile.listFiles();
      ArrayList<CatalogType> result = new ArrayList<CatalogType>();
      for (int i = 0; i < files.length; i++) {
            result.add(new CatalogType(files[i]));
            // go recursively through sub directories
            if (flag.equals("inside") && files[i].isDirectory()) {
                  result.addAll(getFiles(new CatalogType(
                        files[i].getAbsolutePath()), "inside"));
            }
      }
      return result;
   }
   return new ArrayList<CatalogType>(); // return empty list
}

public List<CatalogType> getFiles(CatalogType dirPath) {
      return getFiles(dirPath, "in"); // default to non-recursive
}

public boolean moveFile(CatalogType fromPath, CatalogType toPath) {
      return copyFile(fromPath, toPath) && deleteFile(fromPath);
}
}
```