CLAW

Columbia Language for Altering Waveforms

Shloke Mittal
Jeffrey Polanco
Vivek Ramdev
Bill Wang

Contents

Chapter 1 4
1.1 Introduction 4
CLAW 4
Columbia Language for Altering Waveforms 4
1.2 Background 4
1.2.1 Introduction to Sound/WAVE 4
1.2.2 Introduction to the Java Sound API 5
1.3 Goals of our language: 6
1.3.1 Powerful 6
1.3.2 Practical 6
1.3.3 Simple 7
1.3.4 Portable 7
1.3.5 Efficient 7
Chapter 2 8
Tutorial 8
2.1 The Block 8
2.2 The OUT Block 8
2.3 The .claw File Structure 9
Chapter 3 11
Reference Manual 11
3.1 Lexical Conventions 11
3.1.1 Whitespace 11
3.1.2 Comments 11
3.1.3 Identifiers 11
3.1.4 Separators 11
3.1.5 Keywords 12
3.1.6 Constants 12
3.2 Description of Reserved Words 13
3.3 Grammar 14
Chapter 4 15
Project Plan 15
4.1 Process 15
4.2 Roles and Responsibilities 15
4.3 Code Style Conventions 15
4.3.1 General Principles 16
4.3.2 Line Spacing 16
4.4 Project Timeline 16
Chapter 5 17
Architecture 17
5.1 Program flow 17
5.1.1 Lexer, Parser, and Tree Parser 17
5.1.2 Java Backend 19

5.2 Wave Format 26
5.2.1 CLAW Modifications to Wave Files 27
Chapter 6 29
Test Plan 29
6.1 ANTLR Testing 29
6.2 Java Testing 30
6.2.1 White Box Examples 31
6.2.2 Regression and Black Box Testing Examples 31
Chapter 7 33
Lessons Learned 33
Chapter 8 34
Appendix: Code Listing 34

Chapter 1

1.1 Introduction

CLAW

Columbia Language for Altering Waveforms

CLAW is a programming language that allows the user to easily modify .wav samples and mix them together to create personalized music tracks. CLAW implements a language of simple commands that enables the everyday user to mix together .wav files and manipulate them in a variety of ways. The user can also add effects such as reverb and fade, and then synchronize the samples together and output them as a single .wav file. Likewise, the volume, sample rate, and balance of a sound file can be altered according to the user's preference. CLAW's simple syntax, in combination with its implementation in Java, makes it a quick and efficient way for novice programmers to turn their electronic beats into masterpieces.

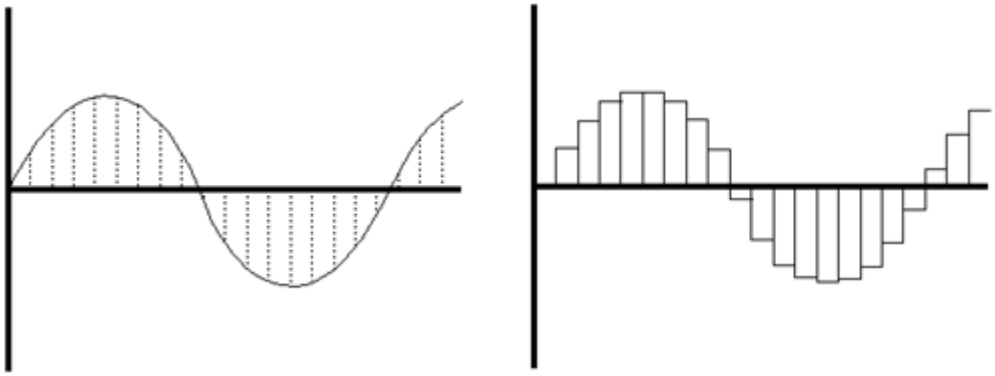
1.2 Background

1.2.1 Introduction to Sound/WAVE

For as long as there has been recorded music, there have been ways to combine instruments and songs to create new sounds and compilations of music. In the past, these methods have been analog. The simple four-track recorder allowed the music enthusiast to use audio tapes to record sounds on four separate tracks, tweak the sound, and combine them into one track for the finished product. Another way that musicians have combined music is by using a mixer that uses vinyl records. This is similar to what you would see a DJ using in a club.

All of these mixers have similar characteristics. They allow someone to take music from separate sources, modify them, and combine them again to create a new sound. These are the goals of CLAW. However, CLAW uses digital audio files stored on your computer, for additional clarity and precision of sound. CLAW takes one or multiple audio samples as inputs, allows you to modify them, and outputs a single .wav file that combines it all.

WAVE has been a standard for digital music for many years. It was developed jointly by Microsoft and IBM for use on PCs, but has become a widely accepted standard that is available on almost all platforms. It can also be played back in all standard web browsers. The incorporation of .wav files into CLAW means that you can rest assured that the code that you write on one machine will work on another.



An analog signal and its digital interpretation

1.2.2 Introduction to the Java Sound API

The backbone implementation of the CLAW language is that of the Java Sound API, which provides the lowest level of sound support on the Java platform. The Java Sound API does not include sophisticated sound editors or graphical tools, but it provides capabilities upon which such programs as CLAW can be built. It emphasizes low-level control beyond that commonly expected by the end user.

The Java Sound API includes support for both digital audio and MIDI data. These two major modules of functionality are provided in separate packages:

- `javax.sound.sampled`

  This package specifies interfaces for capture, mixing, and playback of digital (sampled) audio.

- `javax.sound.midi`

  This package provides interfaces for MIDI synthesis, sequencing, and event transport.

CLAW does not support midi synthesis, and thus the only relevant package used is the javax.sound.sampled.

The `javax.sound.sampled` package is fundamentally concerned with audio transport—it focuses on playback and capture. The main task that the Java Sound API addresses is how to move bytes of formatted audio data into and out of the system. This task involves opening audio input and output devices and managing buffers that get filled with real-time sound data. It can also involve mixing multiple streams of audio into one stream.

To support this focus on basic audio input and output, the Java Sound API provides methods for converting between various audio data formats, and for reading and writing common types of sound files.

A file format specifies the structure of a sound file, including not only the format of the raw audio data in the file, but also other information that can be stored in the file. The sound file CLAW deals with is the WAVE (WAV, as discussed above). The different types of sound file have different structures. For example, they might have a different arrangement of data in the file's "header." A header contains descriptive information that typically precedes the file's actual audio samples, although some file formats allow successive chunks of descriptive and audio data. The header includes a specification of the data format that was used for storing the audio in the sound file. Any of these types of sound file can contain various data formats (although usually there is only one data format within a given file), and the same data format can be used in files that have different file formats.

In the Java Sound API, a file format is represented by an `AudioFileFormat` object, which contains:

- The file type (WAVE, AIFF, etc.)
- The file's length in bytes
- The length, in frames, of the audio data contained in the file
- An AudioFormat object that specifies the data format of the audio data contained in the file

It is this foundation which the CLAW language has in its framework. Effective, potent, and easy to use, CLAW is an easy, effective way to harness the power of the Java Sound API.

1.3 Goals of our language:

1.3.1 Powerful

Designed to be able to perform most of the functions found on your standard music mixer, CLAW can simulate a wide range of effects from adding in fades and other special effects to being able to synchronize two different tracks (such as a baseline and a melody) into one.

1.3.2 Practical

Because music is something which everyone can enjoy, CLAW is something for which everyone can find a use. It is not designed with a specific, technical audience in mind.

1.3.3 Simple

CLAW isn't a language designed for programmers; it's a language designed for musicians. Using very basic English, each command is unambiguous in its function. This means everyday people, with a little music background and a little bit of training, can understand, and even write code, to mix and compose their own music.

1.3.4 Portable

The CLAW compiler takes in the user-generated code and converts it into Java code. This Java code is then run through the standard Java compiler to create the final version, and because Java is able to run in many different environments, this will allow CLAW to run in many different environments as well.

1.3.5 Efficient

Because we are not creating our own compiler but instead using the preexisting Java one, compilation will be very efficient and little time will be wasted with extraneous tasks. Running the code will be fast as well due to the design and implementation of Java.

Chapter 2

Tutorial

2.1 The Block

The most fundamental unit in CLAW is the block. Blocks are where the user will define a file to be modified and describe the modifications to happen to it. CLAW accepts one file per block This allows the user to reuse blocks in other .claw files and gives the language expandability. A block is defined by a name which can be a combination of numbers, letters, and the underscore character, and followed by an open brace. The only other requirement for a block is that it contain the name of the file to be modified and then a closing brace. Therefore, the most simple block that CLAW accepts looks like:

```
block1{
     FILENAME=thefile.wav;
}
```

However, within the block, we can define a number of alterations that will be applied to the file. In this next example, we will modify the wav so that it plays at a quieter volume and has a fade-in effect.

```
block1{
     FILENAME=thefile.wav;
     VOLUME=0.7;
     FADE_IN;
}
```

Do not forget that each line must be terminated with a semicolon and that each decimal numerical value must start with a 0. For example, CLAW will accept 0.7, not .7.

2.2 The OUT Block

The next piece of the CLAW puzzle is the OUT block. Each .claw file must have an OUT block as the last block in the file. It is in this block that the user can define which of the previous blocks to mix, and also apply transformations that will occur to the overall file, which is the product of altering and mixing the first blocks. For example, the user has the option of putting an effect such as a fade_out in the OUT block instead of adding to each previous block. The OUT block must begin with the word OUT and then lists the blocks to mix separated by a plus sign. An example of an out block is:

```
OUT block1 + block2{
     FILENAME=mixed.wav;
```

```
     FADE_OUT;
     REVERB garage;
}
```

2.3 The .claw File Structure

Each .claw file must begin with the name of the file minus the .claw extension. It must then wrap the file in curly braces. After the opening brace, the user can then input one, or up to sixty-four blocks of input and one OUT block. A skeleton example of the file example.claw would look like:

```
example{

     block1{
          FILENAME = …;
          …;
     }

     block2{
          FILENAME = …;
          …;
     }
     …
     OUT block1 + … (
          FILENAME = …;
          …;
     }
}
```

It is also a convention to name each CLAW file with a .claw extension. The compiler will not understand the file otherwise and will give a usage error if the file has any other extension.

2.4 An Everyday Example

Here is an everyday example. This file is called rockbeat.claw and will mix a variety of different parts together. There will be a detailed explanation at the end.

```
rockbeat{

     bass{
          FILENAME = low.wav;
          VOLUME = 0.7;
          FADE_IN;
     }

     rhythm{
          FILENAME = rhythm.wav;
          REVERB garage;
          BALANCE = -1;
     }

     solopart{
          FILENAME = solo.wav;
          BALANCE = 1;
     }

     drum{
          FILENAME = drums.wav;
     }

     OUT bass + rhythm + solopart{
          FILENAME = rockbeat.wav;
          FADE_OUT;
     }
}
```

This .claw file takes a bass, rhythm, and solo part and mixes them together. Notice that while a drum part exists in the file, it does not get mixed into the final product because it is not included in the OUT block. Many different filters and alterations are applied to each of the files to be mixed, and the output is a new and original tune.

Chapter 3

Reference Manual

3.1 Lexical Conventions

The types of tokens we will be dealing with are as follows: identifiers, separators, keywords, and constants. These tokens will be delineated by whitespace as defined below.

3.1.1 Whitespace

Whitespace will separate tokens and is defined as any blank (space), tab, newline, or block of comments. Whitespace will be ignored by the parser as anything but a separator.

3.1.2 Comments

Comments are defined as any block of text between the character **/*** and ***/**. As defined above, comments are considered whitespace and are therefore only count as separators to the compiler.

3.1.3 Identifiers

Identifiers are sequences of letters, numbers, and the underscore character (_). The first characters must be a letter, and the following may be either a number, letter, or underscore. Case matters, so an uppercase letter is different than its lowercase counterpart. Identifiers can be of any length.

3.1.4 Separators

The following characters will be used as separators in CLAW:

{ } ;

In addition, whitespace as defined above will separate tokens.

3.1.5 Keywords

The following words are defined as keywords and are reserved:

```
filename          fade_in
balance           fade_out
volume            reverb
sample_rate
```

3.1.6 Constants

There are few different types of constants in CLAW.

Filename string

The first attribute will be for the filename, which will be a string of characters.

Equals Operators

The next will be for any type of operation that expects a single value, and this will be represented by a float.

Enumerated Operators

Operations such as REVERB have a predefined set of values that they can take, and the compiler will make sure that the option given after an enumerated operator matches an expected option.

Standard Operators

The operations FADE_IN and FADE_OUT take no value. They are static in their function, and therefore are simply written in a file block and followed by a semicolon.

3.2 Description of Reserved Words

FILENAME defines the file to be modified. It must have the extension .wav and be located in the same directory as the rest of the CLAW package.

BALANCE takes in a float value of -1, 1, or 0. A value of 0 indicates equal sound to both channels, -1 indicates full left channel, and 1 indicates full right channel.

FADE_OUT linearly decreases the volume of the last 5 seconds of the wave file to smoothly transition the sample from its indicated volume to silence. It takes no arguments.

FADE_IN linearly increases the volume of the first 5 seconds of the wave file to smoothly transition from silence to the indicated or default volume. It takes no arguments.

REVERB takes in one of five possible arguments (cavern, dungeon, garage, acoustic_lab, or closet). Depending on the environment specified through the argument, a predetermined set of commands will be automatically applied to simulate that environment.

VOLUME takes in a positive float value. Values between 0 and 1 will lessen the volume by a fraction of the current volume. A zero value will effectively mute the track, which would be as easily accomplished by leaving it out of the OUT block. Values greater than 1 will increase the volume. Be careful with this, as clipping will occur if the volume is set too high.

SAMPLE_RATE will be a float value used as a multiplier to modify the frequency of the wave sample. A value greater than 1 will speed up the sample while values lower than 1 will slow it down. SAMPLE_RATE is not an exact science; it can lead to unpredictable outputs.

3.3 Grammar

num → (0…9)

letter → (a…z | A…Z | _)

float → *(‘-‘)? num*$^{+}$ (‘.’*num*+)?

iname → *letter*(*letter* | *num*) *

block → ‘{‘ *FILENAME filename ‘;’ attributes* ‘}’

block1 → *iname block*

attributes → (*attr1* | *attr2* | *attr3*) *

attr1 → *eq_ops* ‘=’ *float ‘;’*

attr2 → *(“FADE_IN”|”FADE_OUT”) ‘;’*

attr3 → ‘REVERB’ *reverb_type ‘;’*

reverb_type → (‘cavern’ | ‘dungeon’ | ‘garage’ | ‘acoustic_lab’ | ‘closet’)

filename → *iname ’.wav’*

eq_ops → (‘BALANCE’ | ‘SAMPLE_RATE’ | ‘VOLUME’)

std_ops → (‘FADE_IN’ | ‘FADE OUT’)

block2 → *OUT: iname* (‘+’ *iname*)* *block*

complete → *iname ‘{‘ block1*$^{+}$ (*block2*) ‘}’

Chapter 4

Project Plan

4.1 Process

The first thing we had to do in order to create our language was understand how wav files are mixed in Java. For this, we read up on documentation about the javax.sound.sampled library. After familiarizing ourselves with how source lines are inputted to create a mixed sound file, we examined the various sound effects available at our disposal. From here, we determined that our language would allow the manipulation of these sound effects for different wav files and also the ability to synchronize the different sound waves into a single one. Our goal was to create a simple language that would permit a user to do the aforementioned with relative ease.

After doing the necessary research, the next step was to design the language to our liking. Thus, we came to an agreement as to how the language would look. We created the lexer and parser so that it would understand the inputted file. We made sure to implement error-checking at all stages so as to complement the default error handling performed by ANTLR. After designing our language, we then modified it so that it would parse correctly as a tree structure. From the nodes, we are able to retrieve all of the important information and store them separately, where they are passed to the java code. Our java code interprets this information and creates the individual source lines depending on the attributes that are defined by the user. In order to achieve the effects that we wanted, we had to manipulate the raw data. This is because the javax library is not complete and the only way we could modify the wav files was by altering the bytes. The final step is that the input lines are mixed together and then outputted as a sound wave.

4.2 Roles and Responsibilities

For the most part, the work was divided into two groups: the ANTLR code and the Java code. In the beginning, we all got involved with ANTLR, during the creation of the Lexer and the Parser. We made sure we understood the process behind our compiler and how it was going to work. After that, we worked separately, with Jeffrey and Shloke finishing off the TreeParser, and Viv and Bill working on the code that mixes the wav files. When it came to documentation, we wrote about our specific parts and then looked over eachother's work before putting it all together.

4.3 Code Style Conventions

The purpose of the code style conventions is to stress the proper coding conventions in your CLAW file, so that retains it reusability and efficiency for others that may come

upon it. Proper adherence to the general rules set forth will ensure programs written will be more beneficial to group members and the programmer who wrote the code, as they will be easily understood, and amended.

4.3.1 General Principles

Code should be laid out in a timely and easily understood manner. Refrain from typing blocks all in one line, as a large block will soon look like clutter. Variables block names, and any other user defined elements should be named in a fashion that reflects their purpose. Comments are always appreciated by anyone reviewing or reusing the code, adding concise and exact comments boost a program's robustness.

4.3.2 Line Spacing

Line spacing should be done in a uniform manner. Irregular amounts of spaces, just add unneeded randomness in your file. Tabs may be used, although be forewarned the way they look in your editor may not reflect the way others may view it.

4.4 Project Timeline

January 30	Group formed
February 7	Brainstormed ideas for project – resolved to do CLAW
February 14	Work on White Paper commences
February 18	White Paper submitted
March 7	Went over the basics of ANTLR. Designed the structure of CLAW
March 14	Divided up work for LRM
March 25	Put LRM together
March 27	Submitted LRM
April 12	Researched Java sound. Finalized functions and capabilities of CLAW.
April 19	Started work on Lexer and Parser
May 3	Continued work on Lexer and Parser. Began work on Mixer.
May 5	Completed Lexer and Parser. Testing and debugging performed. Work on Tree Parser begins. Tested Mixer with different wav files.
May 6	Searched javax.sound.sampled library for line manipulations. Tree Parser completed. More testing of compiler
May 7	Realized javax.sound.sampled library does not support all of the capabilities we were hoping to use. Researched another way to achieve this.
May 8	Discovered bit manipulation of raw data for each line. Worked on code to alter wav files.
May 10	Completed ByteReading.java. Integrated everything. Created GUI for sound playback.
May 11	Testing and debugging performed.
May 12	Created file package.

Chapter 5

Architecture

5.1 Program flow

5.1.1 Lexer, Parser, and Tree Parser

In order to run our program, the user's file must be saved with the ".claw" extension and then the following command must be typed:

$java claw <filename>.claw

The claw.java program invokes our lexer and parser, which in turn read the .claw file while making sure that it is valid. The lexer allows for the following characters:

Numbers (from 0-9), Letters (both lower case and upper case), Floats (decimal numbers) Names (words), Comments (//, /* */), Punctuation Marks ('.' ',' ';' ':' '{' '}' '+' '=' ' ')

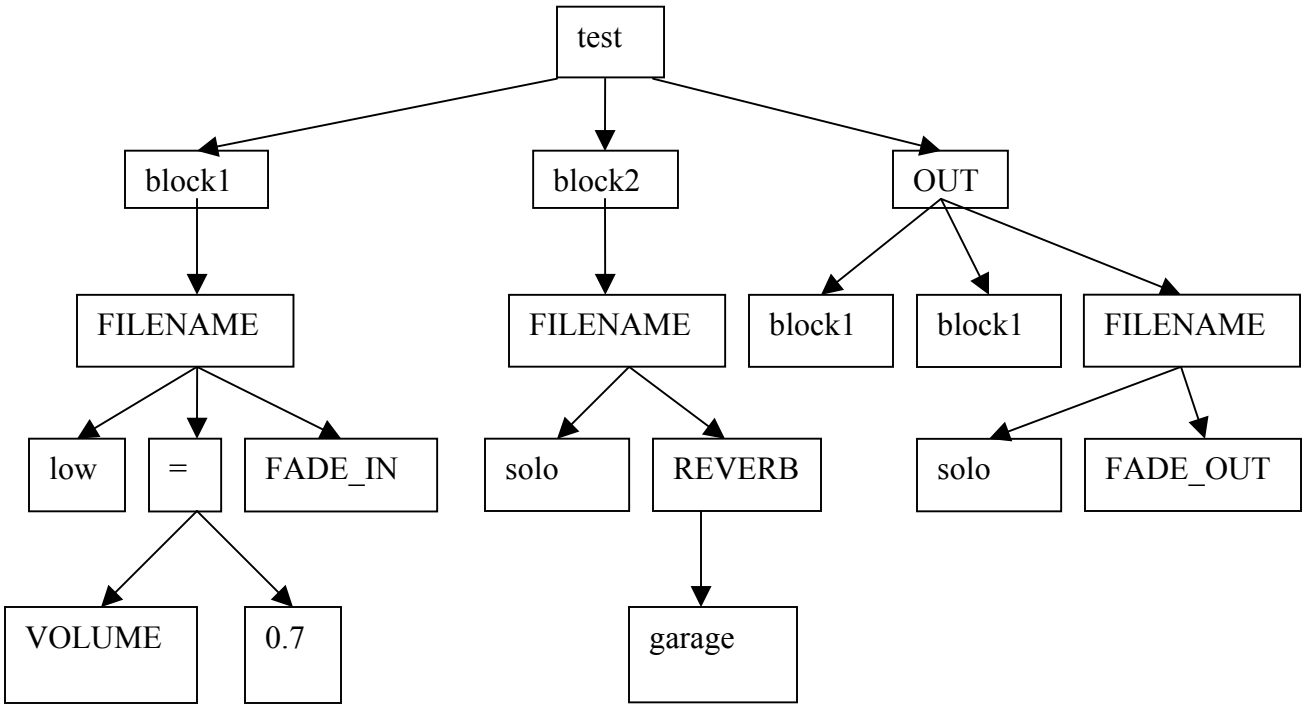
The parser creates the tree while doing basic semantic checks. For example, it verifies that the blocks specified in OUT have been defined previously. It also checks that name of the .claw file matches the header within that file. The tree parser compiles a list of attributes for each block into an InputLine. InputLine is a class that is defined by CLAW, and it holds all of the relevant attribute information for a sound file. Each node of the tree is examined and the values specified by the user are stored as variables in the InputLine. For instance, the InputLine maintains a record of the main characteristics of the wav file (FILENAME, VOLUME, BALANCE, REVERB, etc). If those values are specified in the CLAW file, then the wav is altered accordingly. An array of these InputLines is created when the tree parser goes through the user's file.

The following is an example of how the tree is walked:

```
if (nextNode.getText().equals("=")){
            AST tempNode = nextNode.getFirstChild().getNextSibling();
            if (nextNode.getFirstChild().getText().equals("BALANCE")){
                list[i].balance = Float.parseFloat(tempNode.getText());
                list[i].oneOrOne("balance", list[i].balance);
            }
            else if
(nextNode.getFirstChild().getText().equals("SAMPLE_RATE")){
                list[i].sample_rate =
Float.parseFloat(tempNode.getText());
                list[i].positive("sample rate", list[i].sample_rate);
            }
            else if
(nextNode.getFirstChild().getText().equals("VOLUME")){
                list[i].volume = Float.parseFloat(tempNode.getText());
```

```
            list[i].positive("volume", list[i].volume);
        }
        else{
            System.out.println("error: unrecognized option");
            System.exit(-1);
        }
    }
    else if (nextNode.getText().equals("REVERB"))
        list[i].reverb = nextNode.getFirstChild().getText();
    else if (nextNode.getText().equals("FADE_IN"))
        list[i].fade_in = true;
    else if (nextNode.getText().equals("FADE_OUT"))
        list[i].fade_out = true;
    else{
        System.out.println("error: unrecognized option");
        System.exit(-1);
    }
```

The first attributes searched for in the tree are balance, sample rate, or volume. Because these are the only attributes that use an '=' sign, it is the first node that the TreeParser searches for. Thus, if one is found, the values in the node below it are stored in their proper location in the array. At the same time, it checks to make sure that the numbers indicated by the user are valid. If not, an error is outputted informing the user of this. The next sibling of the '=' sign is reverb, fade_in, or fade_out. If these options are specified in the CLAW file, then they are stored as states in the array. A final check is inserted in case an option is inputted that is not recognized. However, this check is placed just as a convention because the Parser will already notify the user if he inputs an invalid option. The tree parser passes the array of InputLines to the java backend.



A sample abstract syntax tree. It is similar to that outputted by the code in Sec. 2.4.

5.1.2 Java Backend

The backend behind CLAW's manipulation of wave files are implemented all in java using either Sun Microsystems's javax.sound.sampled package, or through the direct byte manipulation of wave files.

The files that are incorporated in this process are listed below:

- InputLine.java — this class acted as an object class that held attributes that represented the user's changes from his/her claw file
- backEnd.java — the main backend java file; acts as the back bone, parsing and calling the java files responsible for byte manipulation and mixing
- byteReading.java — the java file that deals solely with the raw modification of data, to incorporate changes on single files such as volume, reverb, etc.
- AudioConcat.java — the main java file that acts as the backbone for mixing. handles multiple files to be mixed.
- Getopt.java — java file meant to parse input lines, used by AudioConcat
- MixingAudioInput.java — java file that is called by AudioConcat that actually does the mixing of the wave files specified.
- endCLAW.java — GUI that plays the specified file at the end of the mixing process
- Play.java — Used by OpenFileAction to play the specified wav file

The actual process of changing wave files starts when the CLAWInterpreter.java populates an InputLine array and instantiates a backEnd object with the populated InputLine array as a parameter

```
backEnd program = new backEnd(list);
```

```
                                        line 162: CLAWInterpreter.java
```

Once this is passed into backEnd.java the array is circled until the InputLine representing the OUT block is found. CLAW's algorithm in processing files is efficient in that, following the constructs of the language, only block names that are represented in the OUT block will be processed. In this manner, CLAW manages to avoid from processing any wave files in blocks that are not intended to be in the final mix. This saves time, and processor power, that could be utilized elsewhere.

```
      public backEnd(InputLine[] a){
            list = a;
            int i=0,j=0,k=0;

            //cycle until u get the master block
            while(!list[i].master)
                i++;
            master = list[i];
                                        lines 11-18: backEnd.java
```

Once the blocks that need to be processed are recognized, backEnd.java cycles through each of them, instantiating a byteReading object with the appropriate parameters, to apply the changes as defined in the .claw file.

```
      //if one of the input line's name is the same as a block that is
to be mixed
      if (list[i].name.equals(blocks2mix[j])){
                //it is matched add it to the files to be mixed
                String filename_temp = list[i].filename + ".wav";

                files2mix[k]=filename_temp;
                //files2mix[k]=list[i].filename + ".wav";
                k++;
                byteReading bt = new byteReading(filename_temp,
list[i].volume, list[i].sample_rate,
list[i].reverb, list[i].fade_in, list[i].fade_out, list[i].balance);

                                        lines 54-63: backEnd.java
```

The constructor of byteReading is as follows:

```
       public byteReading(String xfilename, float xvolume, float
xsamplerate,
                  String xreverb, boolean xfadein, boolean xfadeout,
                  float xbalance)
                                        line 19 : byteReading.java
```

Since the filename of the wave that needs to be modified is sent as a parameter it is easy for byteReading to open up a File object as defined in the java.io.File package and start its manipulations.

The wave file in consideration always goes through all the byte manipulations of byteReading.java. Standard values are populated in the lexer/parser process so that all parameters will have values when passed into the byteReading class. For example, if the user did not specify any volume change, then VOLUME = 1, is passed to byteReading.java. Standard values such as volume being 1 does not change the file,

since only 0<x<1, 1<x<infinity, values change the file that is outputted. In this manner, all attributes have standard values, so there is no need for intricate checks, or parsing of data.

As mentioned previously all the manipulations: VOLUME, BALANCE, SAMPLE_RATE, REVERB, FADE_IN, FADE_OUT, are done on a byte by byte basis. We deviated from javax.sound.sampled, as it did not support the manipulation of a file, and then saving that file. It only supported the manipulation of a file, and then the playback of that file, something that we were not looking for. An example of the actual byte manipulation that is being done is shown below:

```
/*fading in at the beginning */
if(fadein){
  if(DEBUG) System.out.println("inside fadein");


  for (int i = 44; i<45000 /* #sec * sampleRate */; i++) {
      float multiplier = (float)(i/45000);
      bytes[i] = (byte)(multiplier*bytes[i]);
  }
}
```

lines 132-141 : byteReading.java

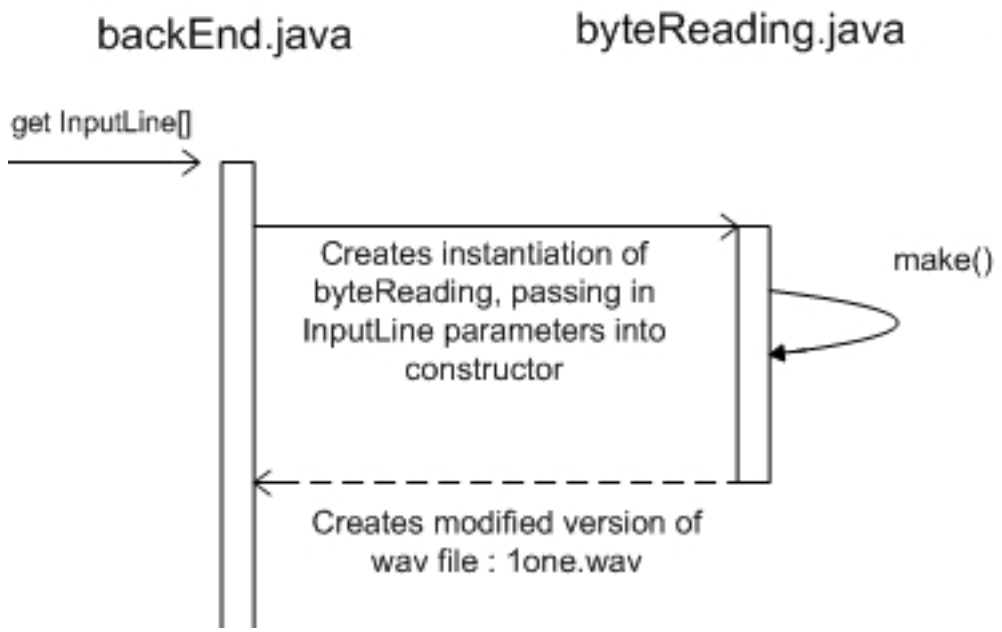
In the example above the bytes array is a byte array that holds the actual bytes of the wave file being modified. More information on how exactly the wave file is defined will follow in the next section.

In other cases, manipulation was still done to bytes, but to the format bytes in the wave file:

```
bytes[24] = (byte)(samplerate * (float)bytes[24]);
bytes[25] = (byte)(samplerate * (float)bytes[25]);
bytes[26] = (byte)(samplerate * (float)bytes[26]);
bytes[27] = (byte)(samplerate * (float)bytes[27]);
```

lines 196-199 : byteReading.java

Such byte manipulations were made possible since certain attributes such as sample_rate being part of the definition of the wave format. The first section can be represented by the following sequence diagram:



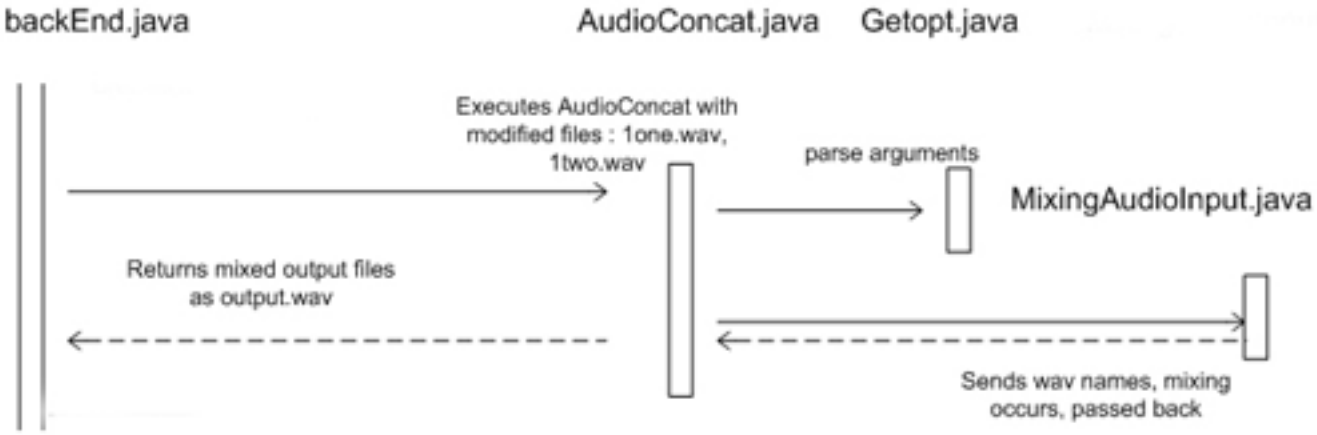
Once all manipulations are finished, byteReading.java creates an output file, that is basically "1"+filename. For example if you had an input file of boo.wav, after the initial manipulations to the single file, a temporary file, 1boo.wav is created. The creation of manipulated single files loops until all files defined in the OUT block of the .claw file are dealt with accordingly. Once this is done, control is then passed back backEnd.java.

With the temporary files ready to be mixed together, backEnd.java calls upon AudioConcat.java, which is open source code that we integrated into out project. The actual call occurs through a runtime execute command with the temporary files as the input to be mixed.

```
//outputting the mix to output.wav
String exec = "java AudioConcat -m -o output.wav ";

//if there are still files to be mixed
while (files2mix[k] != null){

    System.out.print("1"+files2mix[k] +" ");
    //add that to the command line
    exec = exec + "1" + files2mix[k] +" ";
    k++;
}

try{
    Process p = Runtime.getRuntime().exec(exec);
}
catch (Exception e){
    System.err.println(e);
}
```

lines 90-110 : backEnd.java

AudioConcat, since it can take a multiple of input files uses Getopt.java to parse the input line, separating the actual files being mixed between other input.



Once this is done, AudioConcat then instatiates MixingAudioInput.java which uses the javax.sound.sampled, and the synchronize() method, to synch the different AudioInputStreams into one. This is then passed back to AudioConcat, which saves the file as output.wav. This file again, acts as a temporary file in the larger picture of things, since we are not completely done with all the java backend.

Once the mixed output.wav file is created, modifications to it have to be applied, since the user could have easily specified it in the .claw file. Once again byteReading is instantiated, using its other constructor:

```
            public byteReading(String xfilename, float xvolume, float
xsamplerate,
                  String xreverb, boolean xfadein, boolean xfadeout,
                  float xbalance, String xoutputfile)
```

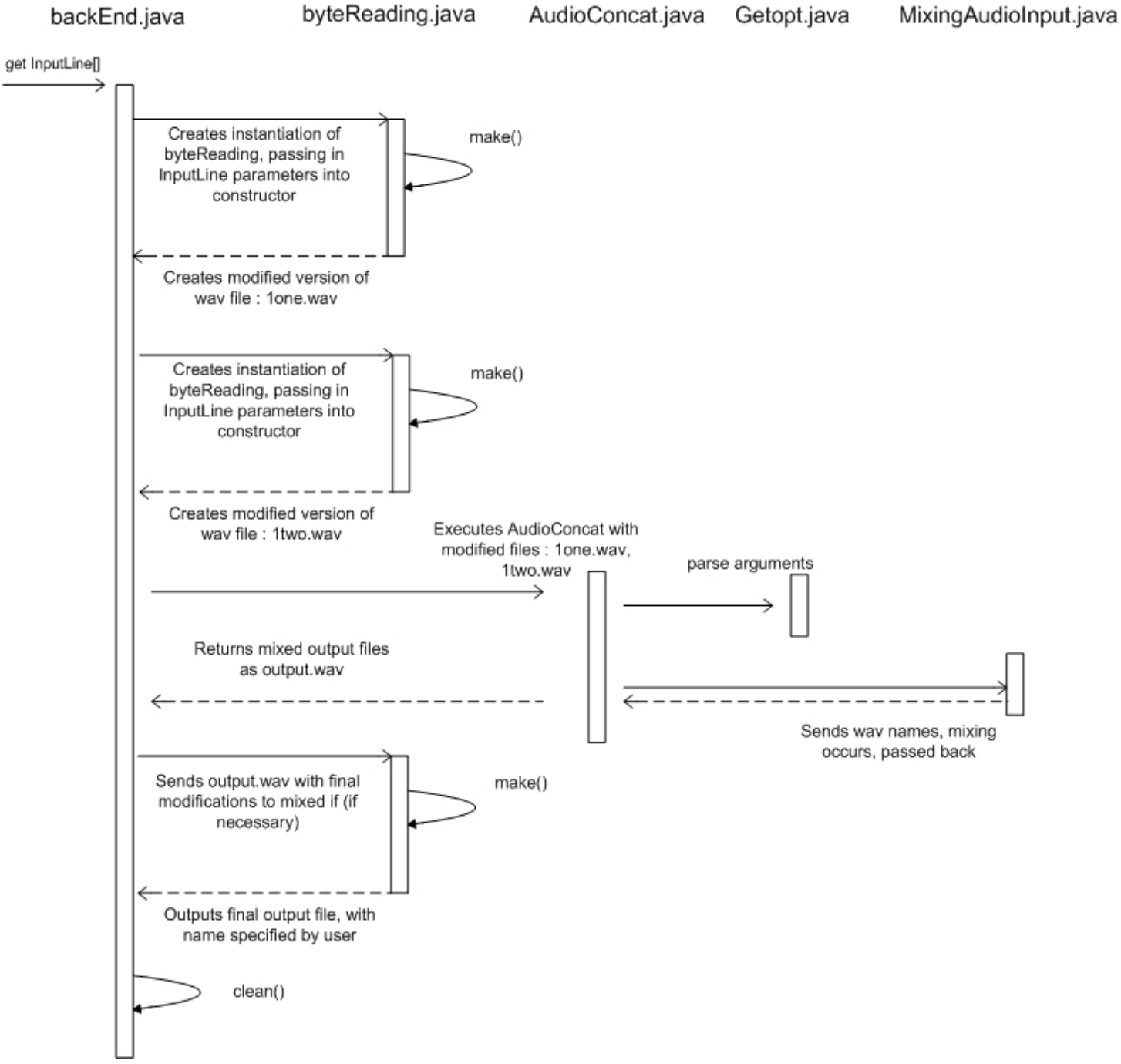
lines 34-36 : byteReading.java

The difference in the constructors lie in the fact that the second one is specified what filename the output file should be named. This is taken from the InputLine that is represents the OUT block in the original .claw file. Once byte modifications are made, the file is saved and control is passed back to the backEnd.java

In a final step, backEnd.java calls endCLAW.java, which is a simple GUI that allows user's to listen to their creations without having to find a wav player. endCLAW.java uses OpenFileAction.java to handle the events of the GUI. The OpenFileAction class, in turn calls upon the simple Play class that plays the selected wav file using javax.sound.sampled.

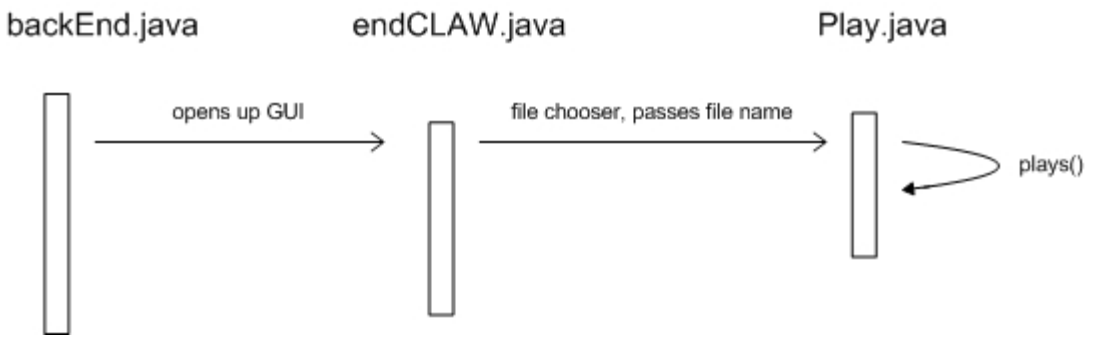
The final sequence diagram is illustrated on the next page.

UML Sequence Diagram:
Example: mix one.wav, two.wav

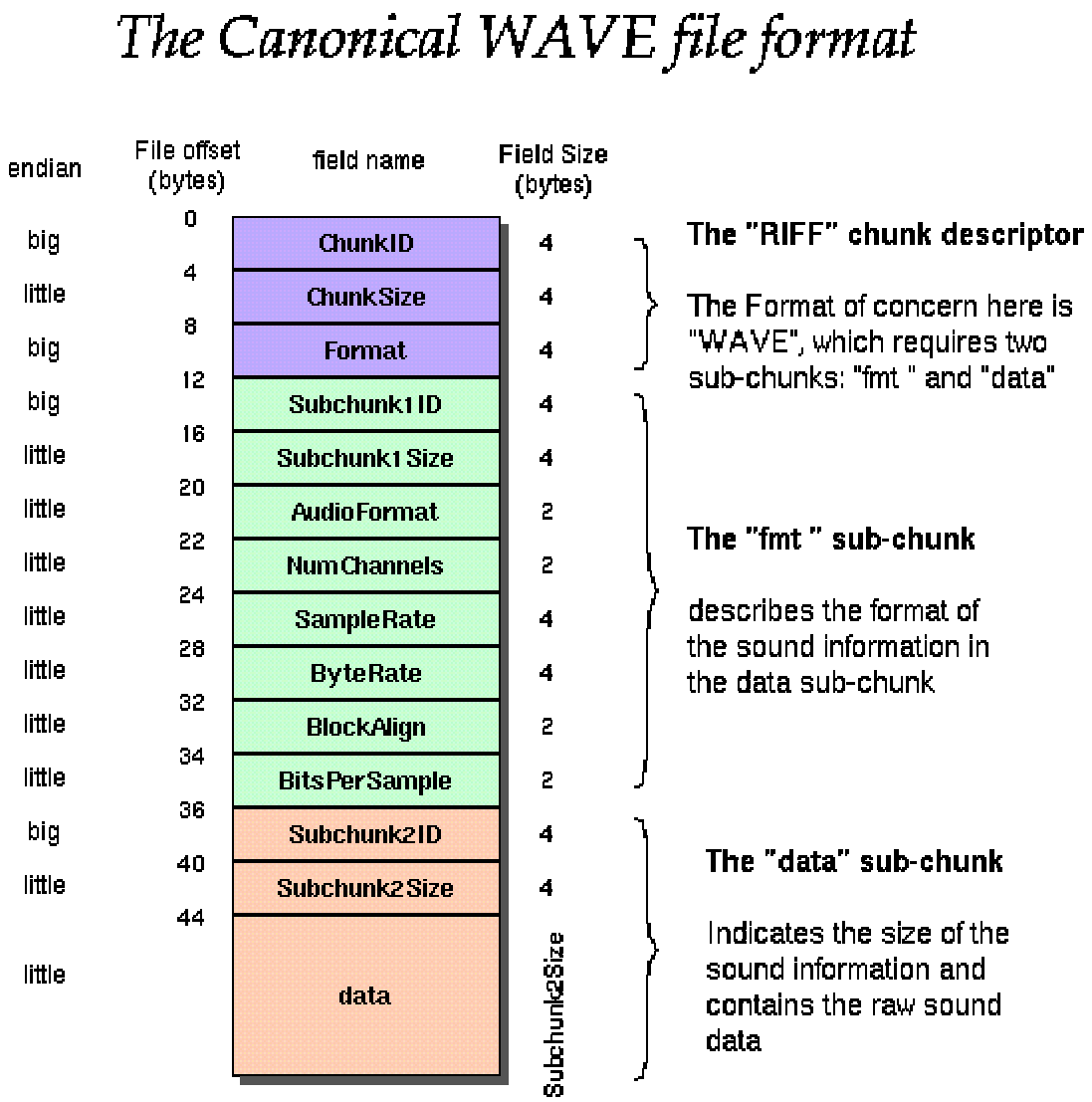


UML Sequence Diagram (continued):

Example: mix one.wav, two.wav



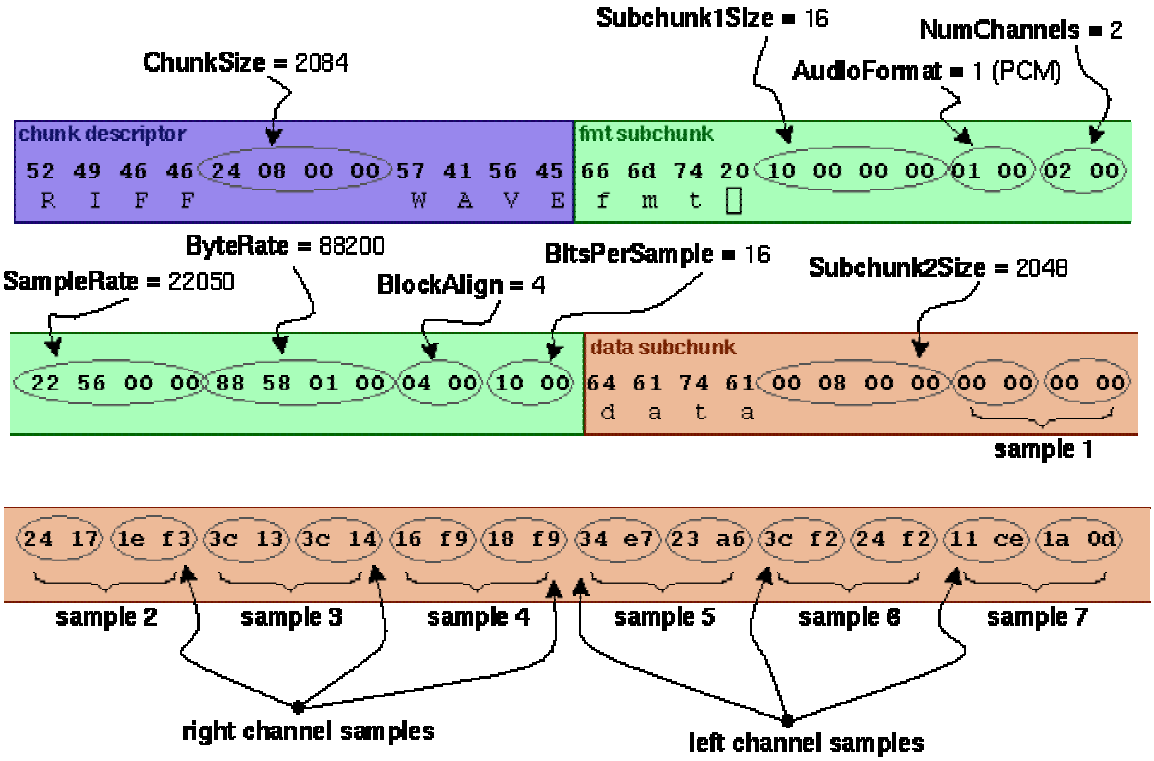
5.2 Wave Format



The standard audio wave file format is a subset of Microsoft's RIFF specification for the storage of multimedia files. More descriptively, it is a combination of a header, used to describe various properties of the file itself, and the rest of the data, which contains the actual information used by applications to generate audio waveforms.

The header is comprised of the first 44 bytes of information in the file. Represented here in hexadecimal form and in either blocks of two or four, these bytes provide the data needed to recognize and play the waveform. For example, the Format bytes will identify the file as a wave format of the RIFF and the ChunkSize will indicate the size of the wave file.

An example of the layout of a wave file can be seen below:



5.2.1 CLAW Modifications to Wave Files

VOLUME – iterating through from the 44th byte to the end of the file, CLAW multiplies each byte of the sample information by the value specified by the language. Example values would be 0.5, 2, 3.4112.

MUTE – iterating through from the 44th byte to the end of the file, each byte of the output is set to 0.

SAMPLE_RATE – bytes 24-27 are multiplied by the new sample rate divided by the old sample rate.

FADE_IN/OUT – given the number of seconds of fade desired, the data bytes are multiplied gradually by float values ranging 0 to 1 or 1 to 0 to simulate the volume increasing/decreasing as time passes.

BALANCE – for a two-channel file, CLAW iterates through the data bytes and multiplies each right channel sample or left channel sample bytes by the value specified in the program. For example, a balance specified by -1.0 would multiply each right channel sample by 0, thereby making all the audio play from the left speaker.

For a one-channel file, CLAW must first change the number of channels to two by modifying byte 22 and halving the current sample rate. Then, an operation to the one mentioned above is performed but is slightly different due to the different format of the data in one-channel files.

REVERB – every other byte is set to zero. This creates an interesting echo similar to that of an empty garage. Unfortunately, only one type of reverb is supported at this time.

Chapter 6

Test Plan

6.1 ANTLR Testing

To test our language we purposely made mistakes in the input file to see if they would be caught properly by the Parser. We also did the same for the various attributes by specifying incorrect values. Below are several examples. The first one shows what happens when the file inputted into CLAW doesn't have the same name as the class header:

```
test1 {
   block1 {
      FILENAME low.wav;
      FADE_IN;
      VOLUME = 0.7;
   }

   OUT : block1 {
      FILENAME mixed.wav;
   }
}

$ make run
/usr/java1.4/bin/java claw test.claw
error: class name must match filename
make: *** [run] Error 255
```

The following demonstrates the result of making a punctuation mistake, such as forgetting a brace, or in this case, leaving out a semi-colon (in block1):

```
test {
   block1 {
      FILENAME low.wav
      FADE_IN;
      VOLUME = 0.7;
   }

   OUT : block1 {
      FILENAME mixed.wav;
   }
}

$ make run
/usr/java1.4/bin/java claw test.claw
error: line 5:9: expecting ';', found 'FADE_IN'
make: *** [run] Error 255
```

Also, if an invalid attribute is specified, the Parser throws an error:

```
test {
   block1 {
```

```
      FILENAME low.wav
      FADE_IN;
      VOLUME = 0.7;
   }

   OUT : block1 {
      FILENAME mixed.wav;
   }
}
$ make run test
/usr/java1.4/bin/java claw test.claw
error: line 6:9: expecting '}', found 'VOL'
make: *** [run] Error 255
```

This final example demonstrates what happens when an invalid filename is specified as part of the output:

```
test {
   block1 {
      FILENAME low.wav;
      FADE_IN;
      VOLUME = 0.7;
   }

block2 {
      FILENAME rhythm.wav;
      REVERB garage;
   }

OUT : block1+block3 {
      FILENAME mixed.wav;
   }
}

$ make run
/usr/java1.4/bin/java claw test.claw
error: block name mismatch
make: *** [run] Error 255
```

6.2 Java Testing

Testing of our Java code was achieved through a mix of white-box testing, black-box testing, and regression testing. Also known as glass box testing and structure testing, white-box uses specific knowledge of the programming code to examine output. During the initial creation of the Java code, line by line, code is tested to ensure that it is being called properly and being executed properly. After the integration of the ANTLR with the Java code, testing moved on to a combination of regression testing, to ensure that previously fixed bugs and formerly working code do not fail as a result of the integration, and black-box testing, to make sure that the code works in general after passing in a .claw file.

6.2.1 White Box Examples

During our white box testing, variables such as strings, filenames, and numerical values are modified to produce errors for the program to catch. For example, we extensively test boundary conditions for many of the numerical values. In such loops as the for loop that changes the volume of the entire wave file located in byteReading.java:

```
for (int i = 44; i < bytes.length; i++) {
      bytes[i] = (bytes)(volume*bytes[i]);
}
```

we try the values of 43, 44 and 45 for i, and i <= bytes.length as the stopping condition (the reason we start at 44 in the first place is that bytes 0 – 43 are part of the header of the wave file). Multiplying byte 43 by a float number would corrupt the wave file depending on what was previously there. If the previous value was 0, then it didn't matter, but if the value was non-zero, then the wave would be unreadable. Starting out at byte 45 would result in the first sound being louder but that's impossible to test considering 22050 bytes are read per second. Furthermore, changing the stopping condition to <= would give an array out of bounds exception. Extreme values were also tested for the float value of volume. Both positive and negative were tested, and it was discovered that clipping would occur at high values.

Values passed through arguments were also tested thoroughly. For example, backEnd.java receives the data from the ANTLR code, parses through it, and sends it to AudioConcat.java and byteReading.java. Errors like incorrect argument types were tested to ensure operation between java files.

We also make extensive use of the DEBUG = true; options in our code. These activate a series of print statement throughout the code to indicate the current process/location of the program as well as to show what values are currently in each variable.

6.2.2 Regression and Black Box Testing Examples

For regression testing, we wanted to make sure no new bugs were introduced after integration. A few bugs were found such as ANTLR passing in floats but the java code expecting an int. Another bug discovered was in the make file. There wasn't enough time elapsed in between the call to create an output file and the call to open that file again. This was corrected by adding a for loop before the call to allow for the output to be finished.

A few black box examples are listed above in the ANTLR testing section. A few more detailed ones were used in this phase, including more effects, more blocks, more mixes, etc. Such an example is:

```
test {
      block1 {
            FILENAME low.wav;
            FADE_IN;
            VOLUME = 0.7;
      }

      block2 {
            FILENAME rhythm.wav;
            REVERB garage;
      }

      block3 {
            FILENAME solo.wav;
      }

      OUT: block1 + block2 {
            FILENAME mixed.wav;
      }
}
```

In this example, we wanted to test some of the basic control options as well as the ability to mix at least 2 files. Adding block3 to the OUT statement would mix 3 files. Adding SAMPLE_RATE to certain blocks would create interesting-sounding waves. Many variations were tested. Most errors are caught first by the tree parser and lexer. The remaining errors would be caught by the Java backend.

Chapter 7

Lessons Learned

ANTLR, it is a very powerful tool, if used properly. It took some time getting familiarized with it. For example, we had trouble deciphering the error messages with our Lexer or Parser that prevented us from proceeding any further on occasion. This was very frustrating at times because we simply could not figure out what ANTLR didn't like about our definitions. However, we were able to progress after we realized our mistakes.

One of the biggest lessons learned from the project is that the javax.sound.sampled package isn't complete. Introduced in JDK version 1.3.1, maybe not all of the controls have been fully implemented. For instance, Mixer.synchronize() seems to have no effect on the lines passed to it. Doing a quick search on the internet returns the result that "synchronization isn't implemented in any known Java Sound implementation." Another such example is DataLine.getLevel().

We also discovered that javax.sound.sampled doesn't actually add effects, such as VOLUME, BALANCE, and SAMPLE_RATE, directly to the wave file. Instead, they are added by manipulating a DataSourceLine retrieved from the wave file. This means, in essence, that the desired effects are made when the sample is played, but the modifications are never saved back to the wave file – DataSourceLines cannot be rewritten. On top of that, doing a Line.getControls() on the DataSourceLine only returns master_gain, mute, apply_reverb, sample rate, and balance as the list of controls available to be modified. Considering that apply_reverb has no noticeable effect, that still leaves out many of the controls described in the Java API.

On the other hand, unlike a DataSourceLine, a TargetSourceLine has the ability to be rewritten into a wave file, however, when Line.getControls() is used on the target, no controls whatsoever are returned. These problems, combined, are what led us to implementing our own byte-modifying code to add effects to the wave file.

Chapter 8

Appendix: Code Listing

```
/* Class claw.java
 *
 *
 * This class starts the interpreter for the CLAW language.
 * The interpreter checks the language and sends the ClAW parameters
 * to the corresponding java tools, which create a wav based on the
 * specifications
 *
 * author: Shloke Mittal
 */


import java.io.*;
import java.lang.*;
import antlr.collections.AST;
import antlr.BaseAST;
import antlr.Token;
import antlr.CommonAST;
import antlr.RecognitionException;
import antlr.TokenStreamException;

class claw{

    public static void main(String args[]){

      boolean DEBUG = false;
      try{
          String filename = args[0];
          checkExt(filename);

          BufferedReader in = new BufferedReader(new
FileReader(filename));
          CLAWlexer lexer = new CLAWlexer(in);
          CLAWparser parser = new CLAWparser(lexer);
          CLAWinterpreter interpreter = new CLAWinterpreter();
          parser.complete(filename);

          if (DEBUG){
            String s = parser.getAST().toStringTree();
            System.out.println(s);
          }

          AST ast = parser.getAST();
          interpreter.go(ast);

      }
      catch(FileNotFoundException e){
          System.err.println("file not found: " +e);
          System.exit(-1);
      }
```

```
      catch(RecognitionException e){
          System.err.println("error: " +e);
          System.exit(-1);
      }
      catch(TokenStreamException e){
          System.err.println("error: " +e);
          System.exit(-1);
      }
    }

    private static void checkExt(String filename){
      char[] temp = filename.toCharArray();
      int i=0;
      String ext = "";
      while ((temp[i]!='.') && (i!=temp.length-1))
          i++;
      if (i==temp.length-1)
          printUsage();
      for ( ; i < temp.length; i++)
          ext += temp[i];
      if (!ext.equals(".claw")){
          printUsage();
      }
    }

    private static void printUsage(){
      System.out.println("\tusage: java claw <filename>.claw");
      System.exit(-1);
    }
}
/* Parser
 *
 *
 * Shloke Mittal
 * Jeff Polanco
 * Vivek Ramdev
 * Bill Wang
 ****************************************/

class CLAWparser extends Parser;

options {

    buildAST=true;
    defaultErrorHandler=  false;
}


block : OBRACE! "FILENAME"^ filename SEMI! attributes CBRACE!;

block1 [int i, String a[]]
    : n:INAME^ block

{   /* Stores the block names in an array, for verification later */
    a[i] = n.getText();
}
;
```

```
attributes : (attr1|attr2|attr3)*;

attr1 : eq_ops EQU^ FLOAT SEMI!;

attr2 : "REVERB"^ reverb_type SEMI!;

attr3 : ("FADE_IN"|"FADE_OUT") SEMI!;

filename : INAME DOT! n:INAME!
{
    //check to make sure that file specified are wavs
    if (!n.getText().equals("wav")){
      System.out.println("error: incorrect file type - files must end
in .wav");
      System.exit(-1);
    }
}
;

eq_ops : ("BALANCE"
        |"SAMPLE_RATE"
        |"VOLUME");

reverb_type : ("cavern"
             |"dungeon"
             |"garage"
             |"acoustic_lab"
             |"closet");


block2 [String a[]]

    : "OUT"^ COL! n:INAME
{
    int i=0;
    boolean works = false;

    //check to make sure that blocks specified in OUT actually exist
    while (a[i] != null){
      if(a[i].equals(n.getText()))
          works = true;
      i++;
    }
    if(!works) {
      System.err.println("error: block name mismatch");
      System.exit(-1);
    }
}

(PLUS! m:INAME)*

{
    if(m != null){
            int j=0;
      boolean works2 = false;
      //continues to check if additional blocks specified in out exist
```

```
      while (a[j] != null){
          if(a[j].equals(m.getText())){
            works2 = true;
          }
          j++;
      }
      if(!works2) {
          System.out.println("error: block name mismatch");
          System.exit(-1);
      }
    }
}

block
;

complete[String f] {
    int i=0;
    String a[] = new String[64];
}
: n:INAME^
{
    //checks to make sure that claw file header matches filename
    char[] s = f.toCharArray();
    int j = 0;
    String fn = "";
    while (s[j] != '.'){
      fn += s[j];
      j++;
    }
    if (!n.getText().equals(fn)){
      System.out.println("error: class name must match filename");
      System.exit(-1);
    }
}
   OBRACE! (block1[i,a]{i++;})+ (block2[a]) CBRACE!
/*{System.out.println("completed");}*/
;




/* Tree parser
 *
 * The TreeParser goes through the tree and creates InputLines for all
blocks.
 * It then passes an array of these InputLines to backEnd.java to
create the
 * wav files.
 *
 * Shloke Mittal
 ****************************************************/
class CLAWinterpreter extends TreeParser;

//{// for emacs formatting purposes

go : #(n:INAME (INAME)+ "OUT")
```

```
{

    boolean DEBUG = false;
    int i = 0;
    InputLine[] list = new InputLine[64];
    AST s = n.getFirstChild();
    //parse through tree, creating InputLines for each block
    while (!s.getText().equals("OUT")){
      list[i] = new InputLine();
      list[i].name = s.getText();
      AST filename = s.getFirstChild().getFirstChild();
      list[i].filename = filename.getText();
      AST nextNode = filename.getNextSibling();
      while(nextNode != null){
          if (nextNode.getText().equals("=")){
            AST tempNode = nextNode.getFirstChild().getNextSibling();
            if (nextNode.getFirstChild().getText().equals("BALANCE")){
                list[i].balance = Float.parseFloat(tempNode.getText());
                list[i].oneOrOne("balance", list[i].balance);
            }
            else if
(nextNode.getFirstChild().getText().equals("SAMPLE_RATE")){
                list[i].sample_rate =
Float.parseFloat(tempNode.getText());
                list[i].positive("sample rate", list[i].sample_rate);
            }
            else if
(nextNode.getFirstChild().getText().equals("VOLUME")){
                list[i].volume = Float.parseFloat(tempNode.getText());
                list[i].positive("volume", list[i].volume);
            }
            else{
                System.out.println("error: unrecognized option");
                System.exit(-1);
            }
          }
          else if (nextNode.getText().equals("REVERB"))
            list[i].reverb = nextNode.getFirstChild().getText();
          else if (nextNode.getText().equals("FADE_IN"))
            list[i].fade_in = true;
          else if (nextNode.getText().equals("FADE_OUT"))
            list[i].fade_out = true;
          else{
            System.out.println("error: unrecognized option");
            System.exit(-1);
          }
          nextNode = nextNode.getNextSibling();
      }
      s = s.getNextSibling();
      i++;
    }
    //now we must be at the OUT, so this is the master block
    list[i] = new InputLine();
    list[i].name = s.getText();
    //set the master tag in InputLine to true
    list[i].master = true;
```

```
    //create a list of those blocks to be mixed, so that we can ignore
those that do not
    AST nextNode = s.getFirstChild();
    int j=0;
    while(!nextNode.getText().equals("FILENAME")){
      list[i].mixList[j] = nextNode.getText();
      j++;
      nextNode = nextNode.getNextSibling();
    }

    nextNode = nextNode.getFirstChild();
    list[i].filename = nextNode.getText();
    nextNode = nextNode.getNextSibling();
    while(nextNode != null){
      if (nextNode.getText().equals("=")){
          AST tempNode = nextNode.getFirstChild().getNextSibling();
          if (nextNode.getFirstChild().getText().equals("BALANCE")){
            list[i].balance = Float.parseFloat(tempNode.getText());
            list[i].oneOrOne("balance", list[i].balance);
          }
          else if
(nextNode.getFirstChild().getText().equals("SAMPLE_RATE")){
            list[i].sample_rate = Float.parseFloat(tempNode.getText());
            list[i].positive("sample rate", list[i].sample_rate);
          }
          else if
(nextNode.getFirstChild().getText().equals("VOLUME")){
            list[i].volume = Float.parseFloat(tempNode.getText());
            list[i].positive("volume", list[i].volume);
          }
          else{
            System.out.println("error: unrecognized option");
            System.exit(-1);
          }
      }
      else if (nextNode.getText().equals("REVERB"))
          list[i].reverb = nextNode.getFirstChild().getText();
      else if (nextNode.getText().equals("FADE_IN"))
          list[i].fade_in = true;
      else if (nextNode.getText().equals("FADE_OUT"))
          list[i].fade_out = true;
      else{
          System.out.println("error: unrecognized option");
          System.exit(-1);
      }
      nextNode = nextNode.getNextSibling();
    }

    //debug mode, print out all the InputLines created
    if (DEBUG){
      i = 0;
      while (list[i] != null){
          list[i].print();
          i++;
      }
    }
```

```
    backEnd program = new backEnd(list);
}
    ;
//}//for emacs formatting purposes




/* Lexer
 *
 *
 * Shloke Mittal
 * Jeff Polanco
 * Vivek Ramdev
 * Bill Wang
************************************************/
class CLAWlexer extends Lexer;

options {
    k=2;
}

protected
NUM : ('0'..'9');

protected
LETTER : ('a'..'z'|'A'..'Z'|'_');

FLOAT
options { paraphrase = "decimal float";}
: ('-')? (NUM)+ ('.' ((NUM)+)? )?;

INAME
options { paraphrase = "name";}
: LETTER (LETTER|NUM)*;

DOT
options { paraphrase = "'.'";}
: '.';

SEMI
options { paraphrase = "';'";}
: ';';

OBRACE
options { paraphrase = "'{'";}
: '{';

CBRACE
options { paraphrase = "'}'";}
: '}';

COMMA
options { paraphrase = "','";}
: ',';
```

```
EQU
options { paraphrase = "'='";}
: '=';

COL
options { paraphrase = "':'";}
: ':';

PLUS
options { paraphrase = "'+'";}
: '+';

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

COMMENT : "/*"

    ( options {greedy=false;} : {LA(2)!='*'}? '/'

      | COMMENT

      | ~('/') )*

    "*/"

{ $setType(Token.SKIP); }
;
// $ANTLR 2.7.2: "claw.g" -> "CLAWlexer.java"$

import java.io.InputStream;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;
import antlr.TokenStreamRecognitionException;
import antlr.CharStreamException;
import antlr.CharStreamIOException;
import antlr.ANTLRException;
import java.io.Reader;
import java.util.Hashtable;
import antlr.CharScanner;
import antlr.InputBuffer;
import antlr.ByteBuffer;
import antlr.CharBuffer;
import antlr.Token;
import antlr.CommonToken;
import antlr.RecognitionException;
import antlr.NoViableAltForCharException;
import antlr.MismatchedCharException;
import antlr.TokenStream;
import antlr.ANTLRHashString;
import antlr.LexerSharedInputState;
```

```
import antlr.collections.impl.BitSet;
import antlr.SemanticException;

public class CLAWlexer extends antlr.CharScanner implements
CLAWparserTokenTypes, TokenStream
 {
public CLAWlexer(InputStream in) {
      this(new ByteBuffer(in));
}
public CLAWlexer(Reader in) {
      this(new CharBuffer(in));
}
public CLAWlexer(InputBuffer ib) {
      this(new LexerSharedInputState(ib));
}
public CLAWlexer(LexerSharedInputState state) {
      super(state);
      caseSensitiveLiterals = true;
      setCaseSensitive(true);
      literals = new Hashtable();
      literals.put(new ANTLRHashString("FILENAME", this), new
Integer(5));
      literals.put(new ANTLRHashString("cavern", this), new
Integer(18));
      literals.put(new ANTLRHashString("closet", this), new
Integer(22));
      literals.put(new ANTLRHashString("FADE_IN", this), new
Integer(12));
      literals.put(new ANTLRHashString("BALANCE", this), new
Integer(15));
      literals.put(new ANTLRHashString("VOLUME", this), new
Integer(17));
      literals.put(new ANTLRHashString("dungeon", this), new
Integer(19));
      literals.put(new ANTLRHashString("acoustic_lab", this), new
Integer(21));
      literals.put(new ANTLRHashString("garage", this), new
Integer(20));
      literals.put(new ANTLRHashString("OUT", this), new Integer(23));
      literals.put(new ANTLRHashString("FADE_OUT", this), new
Integer(13));
      literals.put(new ANTLRHashString("SAMPLE_RATE", this), new
Integer(16));
      literals.put(new ANTLRHashString("REVERB", this), new
Integer(11));
}

public Token nextToken() throws TokenStreamException {
      Token theRetToken=null;
tryAgain:
      for (;;) {
            Token _token = null;
            int _ttype = Token.INVALID_TYPE;
            resetText();
            try {   // for char stream error handling
                  try {   // for lexical error handling
                        switch ( LA(1)) {
```

```
case '-':  case '0':  case '1':  case '2':
case '3':  case '4':  case '5':  case '6':
case '7':  case '8':  case '9':
{
      mFLOAT(true);
      theRetToken=_returnToken;
      break;
}
case 'A':  case 'B':  case 'C':  case 'D':
case 'E':  case 'F':  case 'G':  case 'H':
case 'I':  case 'J':  case 'K':  case 'L':
case 'M':  case 'N':  case 'O':  case 'P':
case 'Q':  case 'R':  case 'S':  case 'T':
case 'U':  case 'V':  case 'W':  case 'X':
case 'Y':  case 'Z':  case '_':  case 'a':
case 'b':  case 'c':  case 'd':  case 'e':
case 'f':  case 'g':  case 'h':  case 'i':
case 'j':  case 'k':  case 'l':  case 'm':
case 'n':  case 'o':  case 'p':  case 'q':
case 'r':  case 's':  case 't':  case 'u':
case 'v':  case 'w':  case 'x':  case 'y':
case 'z':
{
      mINAME(true);
      theRetToken=_returnToken;
      break;
}
case '.':
{
      mDOT(true);
      theRetToken=_returnToken;
      break;
}
case ';':
{
      mSEMI(true);
      theRetToken=_returnToken;
      break;
}
case '{':
{
      mOBRACE(true);
      theRetToken=_returnToken;
      break;
}
case '}':
{
      mCBRACE(true);
      theRetToken=_returnToken;
      break;
}
case ',':
{
      mCOMMA(true);
      theRetToken=_returnToken;
      break;
}
```

```
                        case '=':
                        {
                              mEQU(true);
                              theRetToken=_returnToken;
                              break;
                        }
                        case ':':
                        {
                              mCOL(true);
                              theRetToken=_returnToken;
                              break;
                        }
                        case '+':
                        {
                              mPLUS(true);
                              theRetToken=_returnToken;
                              break;
                        }
                        case '\t':  case '\n':  case '\r':  case ' ':
                        {
                              mWS(true);
                              theRetToken=_returnToken;
                              break;
                        }
                        case '/':
                        {
                              mCOMMENT(true);
                              theRetToken=_returnToken;
                              break;
                        }
                        default:
                        {
                              if (LA(1)==EOF_CHAR) {uponEOF();
_returnToken = makeToken(Token.EOF_TYPE);}
                        else {throw new
NoViableAltForCharException((char)LA(1), getFilename(), getLine(),
getColumn());}
                        }
                        }
                        if ( _returnToken==null ) continue tryAgain; //
found SKIP token
                        _ttype = _returnToken.getType();
                        _ttype = testLiteralsTable(_ttype);
                        _returnToken.setType(_ttype);
                        return _returnToken;
                  }
                  catch (RecognitionException e) {
                        throw new TokenStreamRecognitionException(e);
                  }
            }
            catch (CharStreamException cse) {
                  if ( cse instanceof CharStreamIOException ) {
                        throw new
TokenStreamIOException(((CharStreamIOException)cse).io);
                  }
                  else {
```

```
                        throw new
TokenStreamException(cse.getMessage());
                  }
            }
      }
}

      protected final void mNUM(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = NUM;
            int _saveIndex;

            {
            matchRange('0','9');
            }
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      protected final void mLETTER(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = LETTER;
            int _saveIndex;

            {
            switch ( LA(1)) {
            case 'a':  case 'b':  case 'c':  case 'd':
            case 'e':  case 'f':  case 'g':  case 'h':
            case 'i':  case 'j':  case 'k':  case 'l':
            case 'm':  case 'n':  case 'o':  case 'p':
            case 'q':  case 'r':  case 's':  case 't':
            case 'u':  case 'v':  case 'w':  case 'x':
            case 'y':  case 'z':
            {
                  matchRange('a','z');
                  break;
            }
            case 'A':  case 'B':  case 'C':  case 'D':
            case 'E':  case 'F':  case 'G':  case 'H':
            case 'I':  case 'J':  case 'K':  case 'L':
            case 'M':  case 'N':  case 'O':  case 'P':
            case 'Q':  case 'R':  case 'S':  case 'T':
            case 'U':  case 'V':  case 'W':  case 'X':
            case 'Y':  case 'Z':
            {
                  matchRange('A','Z');
                  break;
            }
            case '_':
            {
                  match('_');
```

```
                  break;
            }
            default:
            {
                  throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
            }
            }
            }
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mFLOAT(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = FLOAT;
            int _saveIndex;

            {
            switch ( LA(1)) {
            case '-':
            {
                  match('-');
                  break;
            }
            case '0':  case '1':  case '2':  case '3':
            case '4':  case '5':  case '6':  case '7':
            case '8':  case '9':
            {
                  break;
            }
            default:
            {
                  throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
            }
            }
            }
            {
            int _cnt33=0;
            _loop33:
            do {
                  if (((LA(1) >= '0' && LA(1) <= '9'))) {
                        mNUM(false);
                  }
                  else {
                        if ( _cnt33>=1 ) { break _loop33; } else {throw
new NoViableAltForCharException((char)LA(1), getFilename(), getLine(),
getColumn());}
                  }

                  _cnt33++;
```

```
            } while (true);
            }
            {
            if ((LA(1)=='.')) {
                  match('.');
                  {
                  if (((LA(1) >= '0' && LA(1) <= '9'))) {
                        {
                        int _cnt37=0;
                        _loop37:
                        do {
                              if (((LA(1) >= '0' && LA(1) <= '9'))) {
                                    mNUM(false);
                              }
                              else {
                                    if ( _cnt37>=1 ) { break _loop37; }
else {throw new NoViableAltForCharException((char)LA(1), getFilename(),
getLine(), getColumn());}
                              }

                              _cnt37++;
                        } while (true);
                        }
                  }
                  else {
                  }

                  }
            }
            else {
            }

            }
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mINAME(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = INAME;
            int _saveIndex;

            mLETTER(false);
            {
            _loop40:
            do {
                  switch ( LA(1)) {
                  case 'A':  case 'B':  case 'C':  case 'D':
                  case 'E':  case 'F':  case 'G':  case 'H':
                  case 'I':  case 'J':  case 'K':  case 'L':
                  case 'M':  case 'N':  case 'O':  case 'P':
                  case 'Q':  case 'R':  case 'S':  case 'T':
```

```
                case 'U':  case 'V':  case 'W':  case 'X':
                case 'Y':  case 'Z':  case '_':  case 'a':
                case 'b':  case 'c':  case 'd':  case 'e':
                case 'f':  case 'g':  case 'h':  case 'i':
                case 'j':  case 'k':  case 'l':  case 'm':
                case 'n':  case 'o':  case 'p':  case 'q':
                case 'r':  case 's':  case 't':  case 'u':
                case 'v':  case 'w':  case 'x':  case 'y':
                case 'z':
                {
                      mLETTER(false);
                      break;
                }
                case '0':  case '1':  case '2':  case '3':
                case '4':  case '5':  case '6':  case '7':
                case '8':  case '9':
                {
                      mNUM(false);
                      break;
                }
                default:
                {
                      break _loop40;
                }
                }
          } while (true);
          }
          if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                _token = makeToken(_ttype);
                _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
          }
          _returnToken = _token;
    }

    public final void mDOT(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
          int _ttype; Token _token=null; int _begin=text.length();
          _ttype = DOT;
          int _saveIndex;

          match('.');
          if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                _token = makeToken(_ttype);
                _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
          }
          _returnToken = _token;
    }

    public final void mSEMI(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
          int _ttype; Token _token=null; int _begin=text.length();
          _ttype = SEMI;
          int _saveIndex;

          match(';');
```

```
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mOBRACE(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = OBRACE;
            int _saveIndex;

            match('{');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mCBRACE(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = CBRACE;
            int _saveIndex;

            match('}');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mCOMMA(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = COMMA;
            int _saveIndex;

            match(',');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mEQU(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = EQU;
```

```
            int _saveIndex;

            match('=');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mCOL(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = COL;
            int _saveIndex;

            match(':');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mPLUS(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = PLUS;
            int _saveIndex;

            match('+');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mWS(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = WS;
            int _saveIndex;

            {
            switch ( LA(1)) {
            case ' ':
            {
                  match(' ');
                  break;
            }
            case '\t':
            {
                  match('\t');
```

```
                  break;
            }
            case '\n':  case '\r':
            {
                  {
                  if ((LA(1)=='\r') && (LA(2)=='\n')) {
                        match("\r\n");
                  }
                  else if ((LA(1)=='\r') && (true)) {
                        match('\r');
                  }
                  else if ((LA(1)=='\n')) {
                        match('\n');
                  }
                  else {
                        throw new
NoViableAltForCharException((char)LA(1), getFilename(), getLine(),
getColumn());
                  }

                  }
                  newline ();
                  break;
            }
            default:
            {
                  throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
            }
            }
            }
            _ttype = Token.SKIP;
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mCOMMENT(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = COMMENT;
            int _saveIndex;

            match("/*");
            {
            _loop55:
            do {
                  // nongreedy exit test
                  if ((LA(1)=='*') && (LA(2)=='/')) break _loop55;
                  if (((LA(1)=='/') &&
(_tokenSet_0.member(LA(2))))&&(LA(2)!='*')) {
                        match('/');
                  }
                  else if ((LA(1)=='/') && (LA(2)=='*')) {
```

```
                        mCOMMENT(false);
                    }
                    else if ((_tokenSet_1.member(LA(1))) &&
(_tokenSet_0.member(LA(2)))) {
                        {
                        match(_tokenSet_1);
                        }
                    }
                    else {
                        break _loop55;
                    }

              } while (true);
              }
              match("*/");
              _ttype = Token.SKIP;
              if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                    _token = makeToken(_ttype);
                    _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
              }
              _returnToken = _token;
        }


        private static final long[] mk_tokenSet_0() {
              long[] data = { 3458760120069006848L, 345876450751230710L,
0L, 0L};
              return data;
        }
        public static final BitSet _tokenSet_0 = new
BitSet(mk_tokenSet_0());
        private static final long[] mk_tokenSet_1() {
              long[] data = { 3458619382580651520L, 345876450751230710L,
0L, 0L};
              return data;
        }
        public static final BitSet _tokenSet_1 = new
BitSet(mk_tokenSet_1());

        }
// $ANTLR 2.7.2: "claw.g" -> "CLAWparser.java"$

import antlr.TokenBuffer;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;
import antlr.ANTLRException;
import antlr.LLkParser;
import antlr.Token;
import antlr.TokenStream;
import antlr.RecognitionException;
import antlr.NoViableAltException;
import antlr.MismatchedTokenException;
import antlr.SemanticException;
import antlr.ParserSharedInputState;
import antlr.collections.impl.BitSet;
import antlr.collections.AST;
```

```
import java.util.Hashtable;
import antlr.ASTFactory;
import antlr.ASTPair;
import antlr.collections.impl.ASTArray;

public class CLAWparser extends antlr.LLkParser       implements
CLAWparserTokenTypes
 {

protected CLAWparser(TokenBuffer tokenBuf, int k) {
  super(tokenBuf,k);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}

public CLAWparser(TokenBuffer tokenBuf) {
  this(tokenBuf,1);
}

protected CLAWparser(TokenStream lexer, int k) {
  super(lexer,k);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}

public CLAWparser(TokenStream lexer) {
  this(lexer,1);
}

public CLAWparser(ParserSharedInputState state) {
  super(state,1);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}

      public final void block() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST block_AST = null;

            match(OBRACE);
            AST tmp4_AST = null;
            tmp4_AST = astFactory.create(LT(1));
            astFactory.makeASTRoot(currentAST, tmp4_AST);
            match(LITERAL_FILENAME);
            filename();
            astFactory.addASTChild(currentAST, returnAST);
            match(SEMI);
            attributes();
            astFactory.addASTChild(currentAST, returnAST);
            match(CBRACE);
            block_AST = (AST)currentAST.root;
```

```
            returnAST = block_AST;
      }

      public final void filename() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST filename_AST = null;
            Token  n = null;
            AST n_AST = null;

            AST tmp7_AST = null;
            tmp7_AST = astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp7_AST);
            match(INAME);
            match(DOT);
            n = LT(1);
            n_AST = astFactory.create(n);
            match(INAME);

            //check to make sure that file specified are wavs
            if (!n.getText().equals("wav")){
                  System.out.println("error: incorrect file type -
files must end in .wav");
                  System.exit(-1);
            }

            filename_AST = (AST)currentAST.root;
            returnAST = filename_AST;
      }

      public final void attributes() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST attributes_AST = null;

            {
            _loop5:
            do {
                  switch ( LA(1)) {
                  case LITERAL_BALANCE:
                  case LITERAL_SAMPLE_RATE:
                  case LITERAL_VOLUME:
                  {
                        attr1();
                        astFactory.addASTChild(currentAST, returnAST);
                        break;
                  }
                  case LITERAL_REVERB:
                  {
                        attr2();
                        astFactory.addASTChild(currentAST, returnAST);
                        break;
                  }
```

```
                case LITERAL_FADE_IN:
                case LITERAL_FADE_OUT:
                {
                      attr3();
                      astFactory.addASTChild(currentAST, returnAST);
                      break;
                }
                default:
                {
                      break _loop5;
                }
                }
          } while (true);
          }
          attributes_AST = (AST)currentAST.root;
          returnAST = attributes_AST;
     }

     public final void block1(
          int i, String a[]
     ) throws RecognitionException, TokenStreamException {

          returnAST = null;
          ASTPair currentAST = new ASTPair();
          AST block1_AST = null;
          Token  n = null;
          AST n_AST = null;

          n = LT(1);
          n_AST = astFactory.create(n);
          astFactory.makeASTRoot(currentAST, n_AST);
          match(INAME);
          block();
          astFactory.addASTChild(currentAST, returnAST);
          /* Stores the block names in an array, for verification
later */
          a[i] = n.getText();

          block1_AST = (AST)currentAST.root;
          returnAST = block1_AST;
     }

     public final void attr1() throws RecognitionException,
TokenStreamException {

          returnAST = null;
          ASTPair currentAST = new ASTPair();
          AST attr1_AST = null;

          eq_ops();
          astFactory.addASTChild(currentAST, returnAST);
          AST tmp9_AST = null;
          tmp9_AST = astFactory.create(LT(1));
          astFactory.makeASTRoot(currentAST, tmp9_AST);
          match(EQU);
          AST tmp10_AST = null;
          tmp10_AST = astFactory.create(LT(1));
```

```
            astFactory.addASTChild(currentAST, tmp10_AST);
            match(FLOAT);
            match(SEMI);
            attr1_AST = (AST)currentAST.root;
            returnAST = attr1_AST;
      }

      public final void attr2() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST attr2_AST = null;

            AST tmp12_AST = null;
            tmp12_AST = astFactory.create(LT(1));
            astFactory.makeASTRoot(currentAST, tmp12_AST);
            match(LITERAL_REVERB);
            reverb_type();
            astFactory.addASTChild(currentAST, returnAST);
            match(SEMI);
            attr2_AST = (AST)currentAST.root;
            returnAST = attr2_AST;
      }

      public final void attr3() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST attr3_AST = null;

            {
            switch ( LA(1)) {
            case LITERAL_FADE_IN:
            {
                  AST tmp14_AST = null;
                  tmp14_AST = astFactory.create(LT(1));
                  astFactory.addASTChild(currentAST, tmp14_AST);
                  match(LITERAL_FADE_IN);
                  break;
            }
            case LITERAL_FADE_OUT:
            {
                  AST tmp15_AST = null;
                  tmp15_AST = astFactory.create(LT(1));
                  astFactory.addASTChild(currentAST, tmp15_AST);
                  match(LITERAL_FADE_OUT);
                  break;
            }
            default:
            {
                  throw new NoViableAltException(LT(1), getFilename());
            }
            }
            }
            match(SEMI);
```

```
            attr3_AST = (AST)currentAST.root;
            returnAST = attr3_AST;
      }

      public final void eq_ops() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST eq_ops_AST = null;

            {
            switch ( LA(1)) {
            case LITERAL_BALANCE:
            {
                  AST tmp17_AST = null;
                  tmp17_AST = astFactory.create(LT(1));
                  astFactory.addASTChild(currentAST, tmp17_AST);
                  match(LITERAL_BALANCE);
                  break;
            }
            case LITERAL_SAMPLE_RATE:
            {
                  AST tmp18_AST = null;
                  tmp18_AST = astFactory.create(LT(1));
                  astFactory.addASTChild(currentAST, tmp18_AST);
                  match(LITERAL_SAMPLE_RATE);
                  break;
            }
            case LITERAL_VOLUME:
            {
                  AST tmp19_AST = null;
                  tmp19_AST = astFactory.create(LT(1));
                  astFactory.addASTChild(currentAST, tmp19_AST);
                  match(LITERAL_VOLUME);
                  break;
            }
            default:
            {
                  throw new NoViableAltException(LT(1), getFilename());
            }
            }
            }
            eq_ops_AST = (AST)currentAST.root;
            returnAST = eq_ops_AST;
      }

      public final void reverb_type() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST reverb_type_AST = null;

            {
            switch ( LA(1)) {
            case LITERAL_cavern:
```

```
        {
              AST tmp20_AST = null;
              tmp20_AST = astFactory.create(LT(1));
              astFactory.addASTChild(currentAST, tmp20_AST);
              match(LITERAL_cavern);
              break;
        }
        case LITERAL_dungeon:
        {
              AST tmp21_AST = null;
              tmp21_AST = astFactory.create(LT(1));
              astFactory.addASTChild(currentAST, tmp21_AST);
              match(LITERAL_dungeon);
              break;
        }
        case LITERAL_garage:
        {
              AST tmp22_AST = null;
              tmp22_AST = astFactory.create(LT(1));
              astFactory.addASTChild(currentAST, tmp22_AST);
              match(LITERAL_garage);
              break;
        }
        case LITERAL_acoustic_lab:
        {
              AST tmp23_AST = null;
              tmp23_AST = astFactory.create(LT(1));
              astFactory.addASTChild(currentAST, tmp23_AST);
              match(LITERAL_acoustic_lab);
              break;
        }
        case LITERAL_closet:
        {
              AST tmp24_AST = null;
              tmp24_AST = astFactory.create(LT(1));
              astFactory.addASTChild(currentAST, tmp24_AST);
              match(LITERAL_closet);
              break;
        }
        default:
        {
              throw new NoViableAltException(LT(1), getFilename());
        }
        }
        }
        reverb_type_AST = (AST)currentAST.root;
        returnAST = reverb_type_AST;
}

public final void block2(
        String a[]
) throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST block2_AST = null;
        Token  n = null;
```

```
            AST n_AST = null;
            Token  m = null;
            AST m_AST = null;

            AST tmp25_AST = null;
            tmp25_AST = astFactory.create(LT(1));
            astFactory.makeASTRoot(currentAST, tmp25_AST);
            match(LITERAL_OUT);
            match(COL);
            n = LT(1);
            n_AST = astFactory.create(n);
            astFactory.addASTChild(currentAST, n_AST);
            match(INAME);

            int i=0;
            boolean works = false;

            //check to make sure that blocks specified in OUT actually
exist
            while (a[i] != null){
                  if(a[i].equals(n.getText()))
                      works = true;
                  i++;
            }
            if(!works) {
                  System.err.println("error: block name mismatch");
                  System.exit(-1);
            }

            {
            _loop17:
            do {
                  if ((LA(1)==PLUS)) {
                        match(PLUS);
                        m = LT(1);
                        m_AST = astFactory.create(m);
                        astFactory.addASTChild(currentAST, m_AST);
                        match(INAME);
                  }
                  else {
                        break _loop17;
                  }

            } while (true);
            }

            if(m != null){
                  int j=0;
                  boolean works2 = false;
                  //continues to check if additional blocks specified
in out exist
                  while (a[j] != null){
                      if(a[j].equals(m.getText())){
                        works2 = true;
                      }
                      j++;
                  }
```

```
                if(!works2) {
                    System.out.println("error: block name mismatch");
                    System.exit(-1);
                }
            }

            block();
            astFactory.addASTChild(currentAST, returnAST);
            block2_AST = (AST)currentAST.root;
            returnAST = block2_AST;
      }

      public final void complete(
            String f
      ) throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST complete_AST = null;
            Token  n = null;
            AST n_AST = null;

            int i=0;
            String a[] = new String[64];


            n = LT(1);
            n_AST = astFactory.create(n);
            astFactory.makeASTRoot(currentAST, n_AST);
            match(INAME);

            //checks to make sure that claw file header matches
filename
            char[] s = f.toCharArray();
            int j = 0;
            String fn = "";
            while (s[j] != '.'){
                  fn += s[j];
                  j++;
            }
            if (!n.getText().equals(fn)){
                  System.out.println("error: class name must match
filename");
                  System.exit(-1);
            }

            match(OBRACE);
            {
            int _cnt20=0;
            _loop20:
            do {
                  if ((LA(1)==INAME)) {
                        block1(i,a);
                        astFactory.addASTChild(currentAST, returnAST);
                        i++;
                  }
                  else {
```

```
                        if ( _cnt20>=1 ) { break _loop20; } else {throw
new NoViableAltException(LT(1), getFilename());}
                  }

                  _cnt20++;
            } while (true);
            }
            {
            block2(a);
            astFactory.addASTChild(currentAST, returnAST);
            }
            match(CBRACE);
            complete_AST = (AST)currentAST.root;
            returnAST = complete_AST;
      }


      public static final String[] _tokenNames = {
            "<0>",
            "EOF",
            "<2>",
            "NULL_TREE_LOOKAHEAD",
            "'{'",
            "\"FILENAME\"",
            "';'",
            "'}'",
            "name",
            "'='",
            "decimal float",
            "\"REVERB\"",
            "\"FADE_IN\"",
            "\"FADE_OUT\"",
            "'.'",
            "\"BALANCE\"",
            "\"SAMPLE_RATE\"",
            "\"VOLUME\"",
            "\"cavern\"",
            "\"dungeon\"",
            "\"garage\"",
            "\"acoustic_lab\"",
            "\"closet\"",
            "\"OUT\"",
            "':'",
            "'+'",
            "NUM",
            "LETTER",
            "','",
            "WS",
            "COMMENT"
      };

      protected void buildTokenTypeASTClassMap() {
            tokenTypeToASTClassMap=null;
      };


      }
```

```
// $ANTLR 2.7.2: "claw.g" -> "CLAWlexer.java"$

public interface CLAWparserTokenTypes {
	int EOF = 1;
	int NULL_TREE_LOOKAHEAD = 3;
	int OBRACE = 4;
	int LITERAL_FILENAME = 5;
	int SEMI = 6;
	int CBRACE = 7;
	int INAME = 8;
	int EQU = 9;
	int FLOAT = 10;
	int LITERAL_REVERB = 11;
	int LITERAL_FADE_IN = 12;
	int LITERAL_FADE_OUT = 13;
	int DOT = 14;
	int LITERAL_BALANCE = 15;
	int LITERAL_SAMPLE_RATE = 16;
	int LITERAL_VOLUME = 17;
	int LITERAL_cavern = 18;
	int LITERAL_dungeon = 19;
	int LITERAL_garage = 20;
	int LITERAL_acoustic_lab = 21;
	int LITERAL_closet = 22;
	int LITERAL_OUT = 23;
	int COL = 24;
	int PLUS = 25;
	int NUM = 26;
	int LETTER = 27;
	int COMMA = 28;
	int WS = 29;
	int COMMENT = 30;
}
// $ANTLR 2.7.2: "claw.g" -> "CLAWinterpreter.java"$

import antlr.TreeParser;
import antlr.Token;
import antlr.collections.AST;
import antlr.RecognitionException;
import antlr.ANTLRException;
import antlr.NoViableAltException;
import antlr.MismatchedTokenException;
import antlr.SemanticException;
import antlr.collections.impl.BitSet;
import antlr.ASTPair;
import antlr.collections.impl.ASTArray;


public class CLAWinterpreter extends antlr.TreeParser       implements
CLAWparserTokenTypes
 {
public CLAWinterpreter() {
	tokenNames = _tokenNames;
}

	public final void go(AST _t) throws RecognitionException {
```

```
            AST go_AST_in = (AST)_t;
            AST n = null;

            try {      // for error handling
                  AST __t23 = _t;
                  n = _t==ASTNULL ? null :(AST)_t;
                  match(_t,INAME);
                  _t = _t.getFirstChild();
                  {
                  int _cnt25=0;
                  _loop25:
                  do {
                        if (_t==null) _t=ASTNULL;
                        if ((_t.getType()==INAME)) {
                              AST tmp1_AST_in = (AST)_t;
                              match(_t,INAME);
                              _t = _t.getNextSibling();
                        }
                        else {
                              if ( _cnt25>=1 ) { break _loop25; } else
{throw new NoViableAltException(_t);}
                        }

                        _cnt25++;
                  } while (true);
                  }
                  AST tmp2_AST_in = (AST)_t;
                  match(_t,LITERAL_OUT);
                  _t = _t.getNextSibling();
                  _t = __t23;
                  _t = _t.getNextSibling();


                  boolean DEBUG = false;
                  int i = 0;
                  InputLine[] list = new InputLine[64];
                  AST s = n.getFirstChild();
                  //parse through tree, creating InputLines for each
block
                  while (!s.getText().equals("OUT")){
                        list[i] = new InputLine();
                        list[i].name = s.getText();
                        AST filename =
s.getFirstChild().getFirstChild();
                        list[i].filename = filename.getText();
                        AST nextNode = filename.getNextSibling();
                        while(nextNode != null){
                            if (nextNode.getText().equals("=")){
                              AST tempNode =
nextNode.getFirstChild().getNextSibling();
                              if
(nextNode.getFirstChild().getText().equals("BALANCE")){
                                    list[i].balance =
Float.parseFloat(tempNode.getText());
                                    list[i].oneOrOne("balance",
list[i].balance);
                              }
```

```
                                else if
(nextNode.getFirstChild().getText().equals("SAMPLE_RATE")){
                                    list[i].sample_rate =
Float.parseFloat(tempNode.getText());
                                    list[i].positive("sample rate",
list[i].sample_rate);
                                }
                                else if
(nextNode.getFirstChild().getText().equals("VOLUME")){
                                    list[i].volume =
Float.parseFloat(tempNode.getText());
                                    list[i].positive("volume",
list[i].volume);
                                }
                                else{
                                    System.out.println("error:
unrecognized option");
                                    System.exit(-1);
                                }
                            }
                            else if
(nextNode.getText().equals("REVERB"))
                                list[i].reverb =
nextNode.getFirstChild().getText();
                            else if
(nextNode.getText().equals("FADE_IN"))
                                list[i].fade_in = true;
                            else if
(nextNode.getText().equals("FADE_OUT"))
                                list[i].fade_out = true;
                            else{
                                System.out.println("error: unrecognized
option");
                                System.exit(-1);
                            }
                            nextNode = nextNode.getNextSibling();
                        }
                        s = s.getNextSibling();
                        i++;
                    }
                    //now we must be at the OUT, so this is the master
block
                    list[i] = new InputLine();
                    list[i].name = s.getText();
                    //set the master tag in InputLine to true
                    list[i].master = true;

                    //create a list of those blocks to be mixed, so that
we can ignore those that do not
                    AST nextNode = s.getFirstChild();
                    int j=0;
                    while(!nextNode.getText().equals("FILENAME")){
                        list[i].mixList[j] = nextNode.getText();
                        j++;
                        nextNode = nextNode.getNextSibling();
                    }
```

```
                    nextNode = nextNode.getFirstChild();
                    list[i].filename = nextNode.getText();
                    nextNode = nextNode.getNextSibling();
                    while(nextNode != null){
                          if (nextNode.getText().equals("=")){
                              AST tempNode =
nextNode.getFirstChild().getNextSibling();
                              if
(nextNode.getFirstChild().getText().equals("BALANCE")){
                                list[i].balance =
Float.parseFloat(tempNode.getText());
                                list[i].oneOrOne("balance",
list[i].balance);
                              }
                              else if
(nextNode.getFirstChild().getText().equals("SAMPLE_RATE")){
                                list[i].sample_rate =
Float.parseFloat(tempNode.getText());
                                list[i].positive("sample rate",
list[i].sample_rate);
                              }
                              else if
(nextNode.getFirstChild().getText().equals("VOLUME")){
                                list[i].volume =
Float.parseFloat(tempNode.getText());
                                list[i].positive("volume",
list[i].volume);
                              }
                              else{
                                System.out.println("error: unrecognized
option");
                                System.exit(-1);
                              }
                          }
                          else if (nextNode.getText().equals("REVERB"))
                              list[i].reverb =
nextNode.getFirstChild().getText();
                          else if (nextNode.getText().equals("FADE_IN"))
                              list[i].fade_in = true;
                          else if (nextNode.getText().equals("FADE_OUT"))
                              list[i].fade_out = true;
                          else{
                              System.out.println("error: unrecognized
option");
                              System.exit(-1);
                          }
                          nextNode = nextNode.getNextSibling();
                    }

                    //debug mode, print out all the InputLines created
                    if (DEBUG){
                          i = 0;
                          while (list[i] != null){
                              list[i].print();
                              i++;
                          }
                    }
```

```
                  backEnd program = new backEnd(list);

            }
            catch (RecognitionException ex) {
                  reportError(ex);
                  if (_t!=null) {_t = _t.getNextSibling();}
            }
            _retTree = _t;
      }


      public static final String[] _tokenNames = {
            "<0>",
            "EOF",
            "<2>",
            "NULL_TREE_LOOKAHEAD",
            "'{'",
            "\"FILENAME\"",
            "','",
            "'}'",
            "name",
            "'='",
            "decimal float",
            "\"REVERB\"",
            "\"FADE_IN\"",
            "\"FADE_OUT\"",
            "'.'",
            "\"BALANCE\"",
            "\"SAMPLE_RATE\"",
            "\"VOLUME\"",
            "\"cavern\"",
            "\"dungeon\"",
            "\"garage\"",
            "\"acoustic_lab\"",
            "\"closet\"",
            "\"OUT\"",
            "':'",
            "'+'",
            "NUM",
            "LETTER",
            "' '",
            "WS",
            "COMMENT"
      };

      }

/* Class InputLine.java
 *
 * This Class serves as a constructor. InputLines hold all necessary
 * info for a line defined by each block of the .claw file. It also has
some
 * special variables for the master line, the line which controls the
 * master output. Finally, there is a print function which can be used
to
```

```
 * print out the contents of a given InputLine during debugging and
some
 * value checkers for error checking during tree parsing.
 *
 * author: Shloke Mittal
 */


public class InputLine {

    String name;
    String filename;
    String reverb;
    float balance;
    float sample_rate;
    float volume;
    boolean fade_in;
    boolean fade_out;
    boolean master;
    String[] mixList = new String[64];

    public InputLine(){

      name = "";
      filename = "";
      reverb = "none";
      balance = 0F;
      sample_rate = 1F;
      volume = 1F;
      fade_in = false;
      fade_out = false;
      master = false;
    }

    public void print(){

      String myReverb = this.reverb;
      if(myReverb == null)
          myReverb = "";

      System.out.print(this.name + ", " +
                  this.filename + ", " +
                  myReverb + ", " +
                  this.balance + ", " +
                  this.sample_rate + ", " +
                  this.volume + ", " +
                  this.fade_in + ", " +
                  this.fade_out + ", " +
                  this.master + ", ");
      int i=0;
      while (this.mixList[i] != null){
          System.out.print(this.mixList[i] + " ");
          i++;
      }
      System.out.println();
    }
```

```
    public void oneToOne(String type, float value){
      if ((value < -1) || (value > 1)){
          System.out.println("error: value of "+ type +" must be
between -1 and 1");
          System.exit(-1);
      }
    }

    public void positive(String type, float value){
      if (value < 0){
          System.out.println("error: value of "+ type +" must be
greater than 0");
          System.exit(-1);
      }
    }

    public void oneOrOne(String type, float value){
      if (!((value==-1) || (value==1) || (value==0))){
          System.out.println("error: value of "+ type + "must be either
-1, 0, or 1");
          System.exit(-1);
      }
    }

}
/*
* Vivek Ramdev
*
*/

import java.lang.Runtime;
import java.io.*;

public class backEnd{
    boolean DEBUG = false;
    InputLine[] list;
    String[] files2mix = new String[64];
    String[] blocks2mix;
    InputLine master;

    public backEnd(InputLine[] a){
      list = a;
      int i=0,j=0,k=0;

      //cycle until u get the master block
      while(!list[i].master)
          i++;
      master = list[i];

      if(DEBUG) {
          System.out.println(" ");
          System.out.println("Master Controls in the beginning: ");
          System.out.println("name: " +master.name);
          System.out.println("filename: " +master.filename);
          System.out.println("reverb: " +master.reverb);
          System.out.println("balance: " +master.balance);
          System.out.println("sample_rate: " +master.sample_rate);
```

```
            System.out.println("volume: " +master.volume);
            System.out.println("fade_in: " +master.fade_in);
            System.out.println("fade_out: " +master.fade_out);
            System.out.println("master should be false: "
+master.master);
        }

        //get the list of blocks you have to mix
        blocks2mix = list[i].mixList;
        i=0;


        while(blocks2mix[i] != null){
            System.out.println(blocks2mix[i]);
            i++;
        }


        i=0;

        //if there is still an input line
        while(list[i] != null){
            j=0;

            //if there are still blocks left
            while (blocks2mix[j] != null){

              //if one of the input line's name is the same as a block
that is to be mixed
              if (list[i].name.equals(blocks2mix[j])){

                  //it is matched add it to the files to be mixed
                  String filename_temp = list[i].filename + ".wav";

                  files2mix[k]=filename_temp;
                  //files2mix[k]=list[i].filename + ".wav";
                  k++;
                  byteReading bt = new byteReading(filename_temp,
list[i].volume, list[i].sample_rate, list[i].reverb, list[i].fade_in,
list[i].fade_out, list[i].balance);

                  if(DEBUG){
                    System.out.println(" ");
                    System.out.println("this is what you must deal
with:");
                    System.out.println("name: " +list[i].name);
                    System.out.println("filename: " +list[i].filename);
                    System.out.println("reverb: " +list[i].reverb);
                    System.out.println("balance: " +list[i].balance);
                    System.out.println("sample_rate: "
+list[i].sample_rate);
                    System.out.println("volume: " +list[i].volume);
                    System.out.println("fade_in: " +list[i].fade_in);
                    System.out.println("fade_out: " +list[i].fade_out);
                    System.out.println("master should be false: "
+list[i].master);
                  }
```

```
            }
            j++;
        }
        i++;
    }

    System.out.println("These files need to be modified: ");
    k=0;

    //outputting the mix to output.wav
    String exec = "java AudioConcat -m -o output.wav ";

    //if there are still files to be mixed
    while (files2mix[k] != null){

        System.out.print("1"+files2mix[k] +" ");
        //add that to the command line
        exec = exec + "1" + files2mix[k] +" ";
        k++;
    }
    System.out.println();

    //test out whatis going to be executed.
    if(DEBUG)
        System.out.println(exec);

    try{
        Process p = Runtime.getRuntime().exec(exec);
    }
    catch (Exception e){
        System.err.println(e);
    }


    //empty loop needed for output.wav to be created fully
    for(int h=0; h<10000; h++)
        {
          for(int y =0; y<10000; y++){
          }
        }

    if(DEBUG) {
        System.out.println(" ");
        System.out.println("Master Controls: ");
        System.out.println("name: " +master.name);
        System.out.println("filename: " +master.filename+".wav");
        System.out.println("reverb: " +master.reverb);
        System.out.println("balance: " +master.balance);
        System.out.println("sample_rate: " +master.sample_rate);
        System.out.println("volume: " +master.volume);
        System.out.println("fade_in: " +master.fade_in);
        System.out.println("fade_out: " +master.fade_out);
        System.out.println("master should be false: "
+master.master);
    }
```

```
      byteReading bt = new byteReading("output.wav", master.volume,
master.sample_rate, master.reverb, master.fade_in, master.fade_out,
master.balance, master.filename+".wav");

      try{
          Process p2 = Runtime.getRuntime().exec("java endCLAW
"+master.filename+".wav");
      }
      catch(Exception e){
          System.err.println(e);
      }
    }
}
/*
* Vivek Ramdev, Bill Wang
*/


import java.io.*;
import java.io.FileInputStream;


public class byteReading{
    byte [] bytes;
    byte [] waveheader;
    int wavehead =0;
    String filename;
    float volume;
    float samplerate;
    String reverb;
    boolean fadein;
    boolean fadeout;
    float balance;
    boolean DEBUG = false;
    String outputfile = "none";

      public byteReading(String xfilename, float xvolume, float
xsamplerate,
                  String xreverb, boolean xfadein, boolean xfadeout,
                  float xbalance)
      {
      filename = xfilename;
      volume = xvolume;
      samplerate = xsamplerate;
      reverb = xreverb;
      fadein = xfadein;
      fadeout = xfadeout;
      balance = xbalance;

      make();
      }

      public byteReading(String xfilename, float xvolume, float
xsamplerate,
                  String xreverb, boolean xfadein, boolean xfadeout,
                  float xbalance, String xoutputfile)
      {
```

```
      filename = xfilename;
      volume = xvolume;
      samplerate = xsamplerate;
      reverb = xreverb;
      fadein = xfadein;
      fadeout = xfadeout;
      balance = xbalance;
      outputfile = xoutputfile;

      make();
      }

    public void make(){

      // this output file the input file with a "1"
      //before it . b/c u'll need different output file

      File out;
      byte[] abData = new byte [510];
      FileOutputStream fileout = null;

      try{

          //this is the file we're reading in from
          System.out.println("finding."+filename);
          File file1 = new File(filename);
          System.out.println("found");
          //file that will be outputed after the changed
          if(outputfile.equals("none")){
            out = new File("1"+filename);
            if(DEBUG)System.out.println("default output name");
          }
          else {
            out = new File(outputfile);
            if(DEBUG)System.out.println("output file = "+outputfile);
          }
          //setting up the size of the bytes
          int filesize = (int)file1.length();
          bytes = new byte[filesize];
          waveheader = new byte[44];

          FileInputStream fileinput = new FileInputStream(file1);
          FileInputStream inputstream = new FileInputStream(file1);

          waveheader = new byte[44];
          wavehead = inputstream.read(waveheader);
          System.out.println(wavehead);

          fileinput.read(bytes, 0, bytes.length);

          fileout = new FileOutputStream(out);
          //System.out.println("bytes: 4-7: " + bytes[4] + " " +
bytes[5] + " " + bytes[6] + " " + bytes[7]);


          if(DEBUG) System.out.println("changing volume to:"+volume);
          for(int i=44; i<bytes.length; i++){
```

```
  bytes[i]=(byte)(volume*bytes[i]); //volume/gain

  /*
  bytes[i/2] = bytes[i];
      if (i > (bytes.length/2))
        bytes[i] = 0;

  //increasing sample rate
    */
}



if(reverb.equals("garage")){

  if(DEBUG) System.out.println("inside garage reverb");

  for(int i = 44; i < bytes.length; i++) {
      if( i%4 == 1) {
        bytes[i-1] = bytes[i];
        bytes[i] = -1;
      }
  }

  //ghetto reverb
}



//be careful here with the 100,000, this is based
//on the fact that the file is larger than that
//needs modification


/*fading in at the beginning */
if(fadein){
  if(DEBUG) System.out.println("inside fadein");


  for (int i = 44; i<45000 /* #sec * sampleRate */; i++) {
      float multiplier = (float)(i/45000);
      bytes[i] = (byte)(multiplier*bytes[i]);
  }
}
/*fading out at the end */
if(fadeout){
  if(DEBUG) System.out.println("inside fadeout");


  for(int i = bytes.length-45000; i < bytes.length; i++) {
      float multiplier = (float)((bytes.length-i)/45000);
      bytes[i] = (byte)(multiplier*bytes[i]);
  }
```

```
          }


          /*

          byte doubleLengthArray[] = new byte[filesize*2];
          boolean doubleSize = false;
          for (int i = 0; i < bytes.length*2; i++) {
            if ( i < 44 )
                doubleLengthArray[i] = bytes[i];
            else {
                if ( i%2 == 0 && i/2>44 ) {
                  doubleLengthArray[i] = bytes[i/2];
                }
                else
                  doubleLengthArray[i] = 0;
            }

          }
          // failed attempt to increase sample rate/double size
          */

          for(int b=0; b<44; b++){
            bytes[b] = waveheader[b];
          } // copying the header 44 bytes.



          //changing sample rates
          //read documentation on ".wav" to learn more

          //bytes[22] = samplerate;
          if(DEBUG){
            System.out.println("setting sample rates to:" +
samplerate);
            System.out.println(bytes[24]);
            System.out.println(bytes[25]);
            System.out.println(bytes[26]);
            System.out.println(bytes[27]);

            System.out.println((samplerate * (float)bytes[24]));
            System.out.println((samplerate * (float)bytes[25]));
            System.out.println((samplerate * (float)bytes[26]));
            System.out.println((samplerate * (float)bytes[27]));
          }

          bytes[24] = (byte)(samplerate * (float)bytes[24]);
          bytes[25] = (byte)(samplerate * (float)bytes[25]);
          bytes[26] = (byte)(samplerate * (float)bytes[26]);
          bytes[27] = (byte)(samplerate * (float)bytes[27]);
          //bytes[22] = 2; //MUST BE SET FOR PAN/BALANCE TO WORK
          //bytes[25] = 40; // divide bytes 24/25/26/27 by 2

          //ALTERNATE METHOD FOR INCREASING/DECREASING, MULTIPLY BYTES
24/25/26/27
```

```
    // Balance stuff
    // all right
    if(balance == 1.0){

      if(DEBUG) System.out.println("balance 1, all right");

      bytes[22] = 2;
      bytes[24] = (byte)((float)0.5 * (float)bytes[24]);
      bytes[25] = (byte)((float)0.5 * (float)bytes[25]);
      bytes[26] = (byte)((float)0.5 * (float)bytes[26]);
      bytes[27] = (byte)((float)0.5 * (float)bytes[27]);

      for(int i = 44; i < bytes.length; i++) {
          if( i%4 == 1) {
            bytes[i-1] = bytes[i];
            bytes[i] = -1;
          }
      }

    }

    //all left
    if(balance == -1.0){

      if(DEBUG) System.out.println("balance -1, all left");

      bytes[22] = 2;
      bytes[24] = (byte)((float)0.5 * (float)bytes[24]);
      bytes[25] = (byte)((float)0.5 * (float)bytes[25]);
      bytes[26] = (byte)((float)0.5 * (float)bytes[26]);
      bytes[27] = (byte)((float)0.5 * (float)bytes[27]);

      for(int i = 44; i < bytes.length; i++) {
          if( i%4 == 3) { // 3 = left, 1 = right
            bytes[i-1] = bytes[i];
            bytes[i] = -1;
          }
      }
    }

    /*
    for(int i = 0; i < 100; i++) {
      System.out.println(i  + ": " + bytes[i]);
    }
    */

    fileout.write(bytes,0,bytes.length);
    System.out.println(bytes.length);


}
catch(FileNotFoundException fe){
    System.err.println("error: file not found");
    fe.printStackTrace();
    System.exit(-1);
}
catch(Exception e){
```

```
            System.out.println("no");
            e.printStackTrace();
          }
        } //matches make



        public void toString2(){
          for(int i=0; i<bytes.length; i++)
            System.out.println(bytes[i]);
        }



} //matches class byteReading
/*
* Bill Wang, Vivek Ramdev
*
*/
import java.io.File;
import java.io.IOException;
import javax.sound.sampled.*;
import java.lang.*;

public class Play{

    public static void main(String [] args){

      Mixer mixmaster;
      SourceDataLine source_line = null;
      SourceDataLine source_line2 = null;
      TargetDataLine output_line = null;
      AudioInputStream audioInputStream = null;
      Mixer.Info [] mixInfo;
      //Clip clip1;

      mixInfo = AudioSystem.getMixerInfo();

      if(args.length < 1) {
          System.out.println("Usage: java shloke <filename>+");
          System.exit(-1);
      }

      String filename = args[0];
      File soundFile1 = new File(filename);


      //we need to read the sould file and put it in the audio input
stream
      try{
          audioInputStream =
AudioSystem.getAudioInputStream(soundFile1);
      }
      catch(Exception e){
          System.out.println("error with the audio input stream");
          e.printStackTrace();
```

```
            System.exit(-1);
        }


        AudioFormat audioFormat = audioInputStream.getFormat();
        DataLine.Info info = new DataLine.Info(SourceDataLine.class,
audioFormat, AudioSystem.NOT_SPECIFIED);
        DataLine.Info info_target = new
DataLine.Info(TargetDataLine.class, audioFormat);
        DataLine.Info info_clip = new DataLine.Info(Clip.class,
audioFormat);

        /*
         * So here's the breakdown:
         * (1)AudioSystem->(2)getAudioInputStream->(3)getFormat of that
         * ->(4)use that to create a DataLine.Info ->(5)who's specs
         * can be used to create a source line
         */

        try{
            //output_line = (TargetDataLine)
AudioSystem.getLine(info_target);
            source_line = (SourceDataLine) AudioSystem.getLine(info);
            //clip1 = (Clip)AudioSystem.getLine(info_clip);
            source_line.open(audioFormat);
            System.out.println("DO  IT");
            //clip1.open(audioInputStream);
            //clip1.start();
        }
        catch(LineUnavailableException e){
            System.out.println("the line is not available");

        }
        catch(Exception e){
            System.out.println("line not created properly");
            e.printStackTrace();
        }


        source_line.start();

        int nBytesRead = 0;
        byte[] abData = new byte[12800];
        while (nBytesRead != -1) {
            try {
              nBytesRead = audioInputStream.read(abData, 0,
abData.length);
              int temp [] = new int[abData.length];
              //System.out.println("The value of " + abData.length);
            }
            catch (IOException e) {
              e.printStackTrace();
            }
            if (nBytesRead >= 0) {
              int nBytesWritten = source_line.write(abData, 0,
nBytesRead);
            }
```

```
      }

      source_line.drain();

      System.out.println("done");
      System.exit(-1);

    }

}
/*
 *      AudioConcat.java
 *
 *      This file is part of the Java Sound Examples.
 */

/*
 *  Copyright (c) 1999 - 2001 by Matthias Pfisterer
<Matthias.Pfisterer@web.de>
 *
 *   This program is free software; you can redistribute it and/or
modify
 *   it under the terms of the GNU Library General Public License as
published
 *   by the Free Software Foundation; either version 2 of the License,
or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *   GNU Library General Public License for more details.
 *
 *   You should have received a copy of the GNU Library General Public
 *   License along with this program; if not, write to the Free
Software
 *   Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 *
 */


import  java.io.File;
import  java.io.IOException;

import  java.util.ArrayList;
import  java.util.List;

import  javax.sound.sampled.AudioFileFormat;
import  javax.sound.sampled.AudioFormat;
import  javax.sound.sampled.AudioInputStream;
import  javax.sound.sampled.AudioSystem;
import  javax.sound.sampled.DataLine;
import  javax.sound.sampled.LineUnavailableException;
import  javax.sound.sampled.Mixer;
import  javax.sound.sampled.SourceDataLine;

/*      If the compilation fails because this class is not available,
```

```
        get gnu.getopt from the URL given in the comment below.
*/
//import  gnu.getopt.Getopt;


// TODO: the name AudioConcat is no longer appropriate. There should be
a name that is neutral to concat/mix.

/*      +DocBookXML
        <title>Concatenating or mixing audio files</title>

        <formalpara><title>Purpose</title>
        <para>This program reads multiple audio files and
        writes a single one either
        containing the data of all the other
        files in order (concatenation mode, option <option>-c</option>)
        or containing a mixdown of all the other files
        (mixing mode, option <option>-m</option>).
        For concatenation, the input files must have the same audio
        format. They need not have the same file type.</para>
        </formalpara>

        <formalpara><title>Level</title>
        <para>experienced</para>
        </formalpara>

        <formalpara><title>Usage</title>
        <para>
        <cmdsynopsis>
        <command>java AudioConcat</command>
        <group choice="plain">
        <arg><option>-c</option></arg>
        <arg><option>-m</option></arg>
        </group>
        <arg choice="plain"><option>-o
<replaceable>outputfile</replaceable></option></arg>
        <arg choice="plain"
rep="repeat"><replaceable>inputfile</replaceable></arg>
        </cmdsynopsis>
        </para>
        </formalpara>

        <formalpara><title>Parameters</title>
        <variablelist>
        <varlistentry>
        <term><option>-c</option></term>
        <listitem><para>selects concatenation mode</para></listitem>
        </varlistentry>
        <varlistentry>
        <term><option>-m</option></term>
        <listitem><para>selects mixing mode</para></listitem>
        </varlistentry>
        <varlistentry>
        <term><option>-o
<replaceable>outputfile</replaceable></option></term>
        <listitem><para>The filename of the output
file</para></listitem>
```

```
        </varlistentry>
        <varlistentry>
        <term><replaceable>inputfile</replaceable></term>
        <listitem><para>the name(s) of input file(s)</para></listitem>
        </varlistentry>
        </variablelist>
        </formalpara>

        <formalpara><title>Bugs, limitations</title>
        <para>
        This program is not well-tested. Output is always a WAV
        file. Future versions should be able to convert
        different audio formats to a dedicated target format.
        </para></formalpara>

        <formalpara><title>Source code</title>
        <para>
        <ulink url="AudioConcat.java.html">AudioConcat.java</ulink>,
        <ulink
url="SequenceAudioInputStream.java.html">SequenceAudioInputStream.java<
/ulink>,
        <ulink
url="MixingAudioInputStream.java.html">MixingAudioInputStream.java</uli
nk>,
        <ulink
url="http://www.urbanophile.com/arenn/hacking/download.html">gnu.getopt
.Getopt</ulink>
        </para>
        </formalpara>

-DocBookXML
*/
public class AudioConcat
{
        private static final int        MODE_NONE = 0;
        private static final int        MODE_MIXING = 1;
        private static final int        MODE_CONCATENATION = 2;

        /**     Flag for debugging messages.
         *      If true, some messages are dumped to the console
         *      during operation.
         */
        private static boolean  DEBUG = false;



        public static void main(String[] args)
        {
                /**     Mode of operation.
                        Determines what to do with the input files:
                        either mixing or concatenation.
                 */
                int             nMode = MODE_NONE;
                String          strOutputFilename = null;
                AudioFormat     audioFormat = null;
                List            audioInputStreamList = new ArrayList();
```

```
                // int  nExternalBufferSize =
DEFAULT_EXTERNAL_BUFFER_SIZE;
                // int  nInternalBufferSize =
AudioSystem.NOT_SPECIFIED;


                /*
                 *      Parsing of command-line options takes place...
                 */
                Getopt  g = new Getopt("AudioConcat", args, "hcmo:");
                int     c;
                while ((c = g.getopt()) != -1)
                {
                        switch (c)
                        {
                        case 'h':
                                printUsageAndExit();

                        case 'o':
                                strOutputFilename = g.getOptarg();
                                if (DEBUG)
                                {

System.out.println("AudioConcat.main(): output filename: " +
strOutputFilename);
                                }
                                break;

                        case 'c':
                                nMode = MODE_CONCATENATION;
                                break;

                        case 'm':
                                nMode = MODE_MIXING;
                                break;

                        case 'D':
                                DEBUG = true;
                                break;

                        case '?':
                                printUsageAndExit();

                        default:
                                System.out.println("AudioConcat.main():
getopt() returned " + c);
                                break;
                        }
                }

                /*
                 *      All remaining arguments are assumed to be
filenames of
                 *      soundfiles we want to play.
                 */
                String  strFilename = null;
                for (int i = g.getOptind(); i < args.length; i++)
```

```
                {
                        strFilename = args[i];
                        File    soundFile = new File(strFilename);

                        /*
                         *      We have to read in the sound file.
                         */
                        AudioInputStream        audioInputStream =
null;
                        try
                        {
                                audioInputStream =
AudioSystem.getAudioInputStream(soundFile);
                        }
                        catch (Exception e)
                        {
                                /*
                                 *      In case of an exception, we
dump the exception
                                 *      including the stack trace to
the console output.
                                 *      Then, we exit the program.
                                 */
                                e.printStackTrace();
                                System.exit(1);
                        }
                        AudioFormat     format =
audioInputStream.getFormat();
                        /*
                          The first input file determines the audio
format. This stream's
                          AudioFormat is stored. All other streams are
checked against
                          this format.
                         */
                        if (audioFormat == null)
                        {
                                audioFormat = format;
                                if (DEBUG)
                                {

System.out.println("AudioConcat.main(): format: " + audioFormat);
                                }
                        }
                        else if ( ! audioFormat.matches(format))
                        {
                                // TODO: try to convert
                                System.out.println("AudioConcat.main():
WARNING: AudioFormats don't match");
                                System.out.println("AudioConcat.main():
master format: " + audioFormat);
                                System.out.println("AudioConcat.main():
this format: " + format);
                        }
                        audioInputStreamList.add(audioInputStream);
                }
```

```
                if (audioFormat == null)
                {
                        System.out.println("No input filenames!");
                        printUsageAndExit();
                }
                AudioInputStream        audioInputStream = null;
                switch (nMode)
                {
                case MODE_CONCATENATION:
                        audioInputStream = new
SequenceAudioInputStream(audioFormat, audioInputStreamList);
                        break;

                case MODE_MIXING:
                        audioInputStream = new
MixingAudioInputStream(audioFormat, audioInputStreamList);
                        break;

                default:
                        System.out.println("you have to specify a mode
(either -m or -c).");
                        printUsageAndExit();
                }

                if (strOutputFilename == null)
                {
                        System.out.println("you have to specify an
output filename (using -o <filename>).");
                        printUsageAndExit();
                }
                File    outputFile = new File(strOutputFilename);
                try
                {
                        AudioSystem.write(audioInputStream,
AudioFileFormat.Type.WAVE, outputFile);
                }
                catch (IOException e)
                {
                        e.printStackTrace();
                  //added by Shloke
                  System.exit(-1);
            }

                if (DEBUG)
                {
                        System.out.println("AudioConcat.main(): before
exit");
                }
                System.exit(0);
        }



        private static void printUsageAndExit()
        {
                System.out.println("AudioConcat: usage:");
```

```
                System.out.println("\tjava AudioConcat -c|-m -o
<outputfile> <inputfile> ...");
                System.exit(1);
        }
}



/*** AudioConcat.java ***/
/**************************************************************************
****
/* Getopt.java -- Java port of GNU getopt from glibc 2.0.6
/*
/* Copyright (c) 1987-1997 Free Software Foundation, Inc.
/* Java Port Copyright (c) 1998 by Aaron M. Renn
(arenn@urbanophile.com)
/*
/* This program is free software; you can redistribute it and/or modify
/* it under the terms of the GNU Library General Public License as
published
/* by  the Free Software Foundation; either version 2 of the License or
/* (at your option) any later version.
/*
/* This program is distributed in the hope that it will be useful, but
/* WITHOUT ANY WARRANTY; without even the implied warranty of
/* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
/* GNU Library General Public License for more details.
/*
/* You should have received a copy of the GNU Library General Public
License
/* along with this program; see the file COPYING.LIB.  If not, write to
/* the Free Software Foundation Inc., 59 Temple Place - Suite 330,
/* Boston, MA  02111-1307 USA
/**************************************************************************
****/

//package gnu.getopt;

import java.util.Locale;
import java.util.ResourceBundle;
import java.util.PropertyResourceBundle;
import java.text.MessageFormat;

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

/**
  * This is a Java port of GNU getopt, a class for parsing command line
  * arguments passed to programs.  It it based on the C getopt()
functions
  * in glibc 2.0.6 and should parse options in a 100% compatible
manner.
  * If it does not, that is a bug.  The programmer's interface is also
  * very compatible.
  * <p>
  * To use Getopt, create a Getopt object with a argv array passed to
the
```

```
  * main method, then call the getopt() method in a loop.  It will
return an
  * int that contains the value of the option character parsed from the
  * command line.  When there are no more options to be parsed, it
  * returns -1.
  * <p>
  * A command line option can be defined to take an argument.  If an
  * option has an argument, the value of that argument is stored in an
  * instance variable called optarg, which can be accessed using the
  * getOptarg() method.  If an option that requires an argument is
  * found, but there is no argument present, then an error message is
  * printed. Normally getopt() returns a '?' in this situation, but
  * that can be changed as described below.
  * <p>
  * If an invalid option is encountered, an error message is printed
  * to the standard error and getopt() returns a '?'.  The value of the
  * invalid option encountered is stored in the instance variable
optopt
  * which can be retrieved using the getOptopt() method.  To suppress
  * the printing of error messages for this or any other error, set
  * the value of the opterr instance variable to false using the
  * setOpterr() method.
  * <p>
  * Between calls to getopt(), the instance variable optind is used to
  * keep track of where the object is in the parsing process.  After
all
  * options have been returned, optind is the index in argv of the
first
  * non-option argument.  This variable can be accessed with the
getOptind()
  * method.
  * <p>
  * Note that this object expects command line options to be passed in
the
  * traditional Unix manner.  That is, proceeded by a '-' character.
  * Multiple options can follow the '-'.  For example "-abc" is
equivalent
  * to "-a -b -c".  If an option takes a required argument, the value
  * of the argument can immediately follow the option character or be
  * present in the next argv element.  For example, "-cfoo" and "-c
foo"
  * both represent an option character of 'c' with an argument of "foo"
  * assuming c takes a required argument.  If an option takes an
argument
  * that is not required, then any argument must immediately follow the
  * option character in the same argv element.  For example, if c takes
  * a non-required argument, then "-cfoo" represents option character
'c'
  * with an argument of "foo" while "-c foo" represents the option
  * character 'c' with no argument, and a first non-option argv element
  * of "foo".
  * <p>
  * The user can stop getopt() from scanning any further into a command
line
  * by using the special argument "--" by itself.  For example:
  * "-a -- -d" would return an option character of 'a', then return -1
  * The "--" is discarded and "-d" is pointed to by optind as the first
```

```
  * non-option argv element.
  * <p>
  * Here is a basic example of using Getopt:
  * <p>
  * <pre>
  * Getopt g = new Getopt("testprog", argv, "ab:c::d");
  * //
  * int c;
  * String arg;
  * while ((c = g.getopt()) != -1)
  *   {
  *     switch(c)
  *       {
  *          case 'a':
  *          case 'd':
  *            System.out.print("You picked " + (char)c + "\n");
  *            break;
  *            //
  *          case 'b':
  *          case 'c':
  *            arg = g.getOptarg();
  *            System.out.print("You picked " + (char)c +
  *                             " with an argument of " +
  *                             ((arg != null) ? arg : "null") + "\n");
  *            break;
  *            //
  *          case '?':
  *            break; // getopt() already printed an error
  *            //
  *          default:
  *            System.out.print("getopt() returned " + c + "\n");
  *       }
  *   }
  * </pre>
  * <p>
  * In this example, a new Getopt object is created with three params.
  * The first param is the program name.  This is for printing error
  * messages in the form "program: error message".  In the C version,
this
  * value is taken from argv[0], but in Java the program name is not
passed
  * in that element, thus the need for this parameter.  The second
param is
  * the argument list that was passed to the main() method.  The third
  * param is the list of valid options.  Each character represents a
valid
  * option.  If the character is followed by a single colon, then that
  * option has a required argument.  If the character is followed by
two
  * colons, then that option has an argument that is not required.
  * <p>
  * Note in this example that the value returned from getopt() is cast
to
  * a char prior to printing.  This is required in order to make the
value
  * display correctly as a character instead of an integer.
  * <p>
```

```
  * If the first character in the option string is a colon, for example
  * ":abc::d", then getopt() will return a ':' instead of a '?' when it
  * encounters an option with a missing required argument.  This allows
the
  * caller to distinguish between invalid options and valid options
that
  * are simply incomplete.
  * <p>
  * In the traditional Unix getopt(), -1 is returned when the first
non-option
  * charcter is encountered.  In GNU getopt(), the default behavior is
to
  * allow options to appear anywhere on the command line.  The getopt()
  * method permutes the argument to make it appear to the caller that
all
  * options were at the beginning of the command line, and all non-
options
  * were at the end.  For example, calling getopt() with command line
args
  * of "-a foo bar -d" returns options 'a' and 'd', then sets optind to
  * point to "foo".  The program would read the last two argv elements
as
  * "foo" and "bar", just as if the user had typed "-a -d foo bar".
  * <p>
  * The user can force getopt() to stop scanning the command line with
  * the special argument "--" by itself.  Any elements occuring before
the
  * "--" are scanned and permuted as normal.  Any elements after the "-
-"
  * are returned as is as non-option argv elements.  For example,
  * "foo -a -- bar -d" would return  option 'a' then -1.  optind would
point
  * to "foo", "bar" and "-d" as the non-option argv elements.  The "--"
  * is discarded by getopt().
  * <p>
  * There are two ways this default behavior can be modified.  The
first is
  * to specify traditional Unix getopt() behavior (which is also POSIX
  * behavior) in which scanning stops when the first non-option
argument
  * encountered.  (Thus "-a foo bar -d" would return 'a' as an option
and
  * have "foo", "bar", and "-d" as non-option elements).  The second is
to
  * allow options anywhere, but to return all elements in the order
they
  * occur on the command line.  When a non-option element is
ecountered,
  * an integer 1 is returned and the value of the non-option element is
  * stored in optarg is if it were the argument to that option.  For
  * example, "-a foo -d", returns first 'a', then 1 (with optarg set to
  * "foo") then 'd' then -1.  When this "return in order" functionality
  * is enabled, the only way to stop getopt() from scanning all command
  * line elements is to use the special "--" string by itself as
described
  * above.  An example is "-a foo -b -- bar", which would return 'a',
then
```

```
  * integer 1 with optarg set to "foo", then 'b', then -1.  optind
would
  * then point to "bar" as the first non-option argv element.  The "--"
  * is discarded.
  * <p>
  * The POSIX/traditional behavior is enabled by either setting the
  * property "gnu.posixly_correct" or by putting a '+' sign as the
first
  * character of the option string.  The difference between the two
  * methods is that setting the gnu.posixly_correct property also
forces
  * certain error messages to be displayed in POSIX format.  To enable
  * the "return in order" functionality, put a '-' as the first
character
  * of the option string.  Note that after determining the proper
  * behavior, Getopt strips this leading '+' or '-', meaning that a ':'
  * placed as the second character after one of those two will still
cause
  * getopt() to return a ':' instead of a '?' if a required option
  * argument is missing.
  * <p>
  * In addition to traditional single character options, GNU Getopt
also
  * supports long options.  These are preceeded by a "--" sequence and
  * can be as long as desired.  Long options provide a more user-
friendly
  * way of entering command line options.  For example, in addition to
a
  * "-h" for help, a program could support also "--help".
  * <p>
  * Like short options, long options can also take a required or non-
required
  * argument.  Required arguments can either be specified by placing an
  * equals sign after the option name, then the argument, or by putting
the
  * argument in the next argv element.  For example: "--outputdir=foo"
and
  * "--outputdir foo" both represent an option of "outputdir" with an
  * argument of "foo", assuming that outputdir takes a required
argument.
  * If a long option takes a non-required argument, then the equals
sign
  * form must be used to specify the argument.  In this case,
  * "--outputdir=foo" would represent option outputdir with an argument
of
  * "foo" while "--outputdir foo" would represent the option outputdir
  * with no argument and a first non-option argv element of "foo".
  * <p>
  * Long options can also be specified using a special POSIX argument
  * format (one that I highly discourage).  This form of entry is
  * enabled by placing a "W;" (yes, 'W' then a semi-colon) in the valid
  * option string.  This causes getopt to treat the name following the
  * "-W" as the name of the long option.  For example, "-W
outputdir=foo"
  * would be equivalent to "--outputdir=foo".  The name can immediately
  * follow the "-W" like so: "-Woutputdir=foo".  Option arguments are
```

```
  * handled identically to normal long options.  If a string follows
the
  * "-W" that does not represent a valid long option, then getopt()
returns
  * 'W' and the caller must decide what to do.  Otherwise getopt()
returns
  * a long option value as described below.
  * <p>
  * While long options offer convenience, they can also be tedious to
type
  * in full.  So it is permissible to abbreviate the option name to as
  * few characters as required to uniquely identify it.  If the name
can
  * represent multiple long options, then an error message is printed
and
  * getopt() returns a '?'.
  * <p>
  * If an invalid option is specified or a required option argument is
  * missing, getopt() prints an error and returns a '?' or ':' exactly
  * as for short options.  Note that when an invalid long option is
  * encountered, the optopt variable is set to integer 0 and so cannot
  * be used to identify the incorrect option the user entered.
  * <p>
  * Long options are defined by LongOpt objects.  These objects are
created
  * with a contructor that takes four params: a String representing the
  * object name, a integer specifying what arguments the option takes
  * (the value is one of LongOpt.NO_ARGUMENT,
LongOpt.REQUIRED_ARGUMENT,
  * or LongOpt.OPTIONAL_ARGUMENT), a StringBuffer flag object
(described
  * below), and an integer value (described below).
  * <p>
  * To enable long option parsing, create an array of LongOpt's
representing
  * the legal options and pass it to the Getopt() constructor.
WARNING: If
  * all elements of the array are not populated with LongOpt objects,
the
  * getopt() method will throw a NullPointerException.
  * <p>
  * When getopt() is called and a long option is encountered, one of
two
  * things can be returned.  If the flag field in the LongOpt object
  * representing the long option is non-null, then the integer value
field
  * is stored there and an integer 0 is returned to the caller.  The
val
  * field can then be retrieved from the flag field.  Note that since
the
  * flag field is a StringBuffer, the appropriate String to integer
converions
  * must be performed in order to get the actual int value stored
there.
  * If the flag field in the LongOpt object is null, then the value
field
```

```
  * of the LongOpt is returned.  This can be the character of a short
option.
  * This allows an app to have both a long and short option sequence
  * (say, "-h" and "--help") that do the exact same thing.
  * <p>
  * With long options, there is an alternative method of determining
  * which option was selected.  The method getLongind() will return the
  * the index in the long option array (NOT argv) of the long option
found.
  * So if multiple long options are configured to return the same
value,
  * the application can use getLongind() to distinguish between them.
  * <p>
  * Here is an expanded Getopt example using long options and various
  * techniques described above:
  * <p>
  * <pre>
  * int c;
  * String arg;
  * LongOpt[] longopts = new LongOpt[3];
  * //
  * StringBuffer sb = new StringBuffer();
  * longopts[0] = new LongOpt("help", LongOpt.NO_ARGUMENT, null, 'h');
  * longopts[1] = new LongOpt("outputdir", LongOpt.REQUIRED_ARGUMENT,
sb, 'o');
  * longopts[2] = new LongOpt("maximum", LongOpt.OPTIONAL_ARGUMENT,
null, 2);
  * //
  * Getopt g = new Getopt("testprog", argv, "-:bc::d:hW;", longopts);
  * g.setOpterr(false); // We'll do our own error handling
  * //
  * while ((c = g.getopt()) != -1)
  *   switch (c)
  *     {
  *       case 0:
  *         arg = g.getOptarg();
  *         System.out.println("Got long option with value '" +
  *                            (char)(new
Integer(sb.toString())).intValue()
  *                            + "' with argument " +
  *                            ((arg != null) ? arg : "null"));
  *         break;
  *         //
  *       case 1:
  *         System.out.println("I see you have return in order set and
that " +
  *                            "a non-option argv element was just
found " +
  *                            "with the value '" + g.getOptarg() +
"'");
  *         break;
  *         //
  *       case 2:
  *         arg = g.getOptarg();
  *         System.out.println("I know this, but pretend I didn't");
  *         System.out.println("We picked option " +
  *                            longopts[g.getLongind()].getName() +
```

```
  *                           " with value " +
  *                           ((arg != null) ? arg : "null"));
  *           break;
  *           //
  *         case 'b':
  *           System.out.println("You picked plain old option " +
(char)c);
  *           break;
  *           //
  *         case 'c':
  *         case 'd':
  *           arg = g.getOptarg();
  *           System.out.println("You picked option '" + (char)c +
  *                              "' with argument " +
  *                              ((arg != null) ? arg : "null"));
  *           break;
  *           //
  *         case 'h':
  *           System.out.println("I see you asked for help");
  *           break;
  *           //
  *         case 'W':
  *           System.out.println("Hmmm. You tried a -W with an incorrect
long " +
  *                              "option name");
  *           break;
  *           //
  *         case ':':
  *           System.out.println("Doh! You need an argument for option "
+
  *                              (char)g.getOptopt());
  *           break;
  *           //
  *         case '?':
  *           System.out.println("The option '" + (char)g.getOptopt() +
  *                              "' is not valid");
  *           break;
  *           //
  *         default:
  *           System.out.println("getopt() returned " + c);
  *           break;
  *       }
  * //
  * for (int i = g.getOptind(); i < argv.length ; i++)
  *   System.out.println("Non option argv element: " + argv[i] + "\n");
  * </pre>
  * <p>
  * There is an alternative form of the constructor used for long
options
  * above.  This takes a trailing boolean flag.  If set to false,
Getopt
  * performs identically to the example, but if the boolean flag is
true
  * then long options are allowed to start with a single '-' instead of
  * "--".  If the first character of the option is a valid short option
  * character, then the option is treated as if it were the short
option.
```

```
  * Otherwise it behaves as if the option is a long option.  Note that
  * the name given to this option - long_only - is very counter-
intuitive.
  * It does not cause only long options to be parsed but instead
enables
  * the behavior described above.
  * <p>
  * Note that the functionality and variable names used are driven from
  * the C lib version as this object is a port of the C code, not a
  * new implementation.  This should aid in porting existing C/C++
code,
  * as well as helping programmers familiar with the glibc version to
  * adapt to the Java version even if it seems very non-Java at times.
  * <p>
  * In this release I made all instance variables protected due to
  * overwhelming public demand.  Any code which relied on optarg,
  * opterr, optind, or optopt being public will need to be modified to
  * use the appropriate access methods.
  * <p>
  * Please send all bug reports, requests, and comments to
  * <a href="mailto:arenn@urbanophile.com">arenn@urbanophile.com</a>.
  *
  * @version 1.0.7
  *
  * @author Roland McGrath (roland@gnu.ai.mit.edu)
  * @author Ulrich Drepper (drepper@cygnus.com)
  * @author Aaron M. Renn (arenn@urbanophile.com)
  *
  * @see LongOpt
  */
public class Getopt extends Object
{

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

/*
 * Class Variables
 */

/**
  * Describe how to deal with options that follow non-option ARGV-
elements.
  *
  * If the caller did not specify anything,
  * the default is REQUIRE_ORDER if the property
  * gnu.posixly_correct is defined, PERMUTE otherwise.
  *
  * The special argument `--' forces an end of option-scanning
regardless
  * of the value of `ordering'.  In the case of RETURN_IN_ORDER, only
  * `--' can cause `getopt' to return -1 with `optind' != ARGC.
  *
  * REQUIRE_ORDER means don't recognize them as options;
  * stop option processing when the first non-option is seen.
  * This is what Unix does.
  * This mode of operation is selected by either setting the property
```

```
  * gnu.posixly_correct, or using `+' as the first character
  * of the list of option characters.
  */
protected static final int REQUIRE_ORDER = 1;

/**
  * PERMUTE is the default.  We permute the contents of ARGV as we
scan,
  * so that eventually all the non-options are at the end.  This allows
options
  * to be given in any order, even with programs that were not written
to
  * expect this.
  */
protected static final int PERMUTE = 2;

/**
  * RETURN_IN_ORDER is an option available to programs that were
written
  * to expect options and other ARGV-elements in any order and that
care about
  * the ordering of the two.  We describe each non-option ARGV-element
  * as if it were the argument of an option with character code 1.
  * Using `-' as the first character of the list of option characters
  * selects this mode of operation.
  */
protected static final int RETURN_IN_ORDER = 3;

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

/*
 * Instance Variables
 */

/**
  * For communication from `getopt' to the caller.
  * When `getopt' finds an option that takes an argument,
  * the argument value is returned here.
  * Also, when `ordering' is RETURN_IN_ORDER,
  * each non-option ARGV-element is returned here.
  */
protected String optarg;

/**
  *  Index in ARGV of the next element to be scanned.
  *  This is used for communication to and from the caller
  *  and for communication between successive calls to `getopt'.
  *
  *  On entry to `getopt', zero means this is the first call;
initialize.
  *
  *  When `getopt' returns -1, this is the index of the first of the
  *  non-option elements that the caller should itself scan.
  *
  *  Otherwise, `optind' communicates from one call to the next
  *  how much of ARGV has been scanned so far.
```

```
  */
protected int optind = 0;

/**
  * Callers store false here to inhibit the error message
  * for unrecognized options.
  */
protected boolean opterr = true;

/**
  * When an unrecognized option is encountered, getopt will return a
'?'
  * and store the value of the invalid option here.
  */
protected int optopt = '?';

/**
  * The next char to be scanned in the option-element
  * in which the last option character we returned was found.
  * This allows us to pick up the scan where we left off.
  *
  * If this is zero, or a null string, it means resume the scan
  * by advancing to the next ARGV-element.
  */
protected String nextchar;

/**
  * This is the string describing the valid short options.
  */
protected String optstring;

/**
  * This is an array of LongOpt objects which describ the valid long
  * options.
  */
protected LongOpt[] long_options;

/**
  * This flag determines whether or not we are parsing only long args
  */
protected boolean long_only;

/**
  * Stores the index into the long_options array of the long option
found
  */
protected int longind;

/**
  * The flag determines whether or not we operate in strict POSIX
compliance
  */
protected boolean posixly_correct;

/**
  * A flag which communicates whether or not checkLongOption() did all
  * necessary processing for the current option
```

```
  */
protected boolean longopt_handled;

/**
  * The index of the first non-option in argv[]
  */
protected int first_nonopt = 1;

/**
  * The index of the last non-option in argv[]
  */
protected int last_nonopt = 1;

/**
  * Flag to tell getopt to immediately return -1 the next time it is
  * called.
  */
private boolean endparse = false;

/**
  * Saved argument list passed to the program
  */
protected String[] argv;

/**
  * Determines whether we permute arguments or not
  */
protected int ordering;

/**
  * Name to print as the program name in error messages.  This is
necessary
  * since Java does not place the program name in argv[0]
  */
protected String progname;

/**
  * The localized strings are kept in a separate file
  */
private ResourceBundle _messages = PropertyResourceBundle.getBundle(
                           "gnu/getopt/MessagesBundle",
Locale.getDefault());

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

/*
 * Constructors
 */

/**
  * Construct a basic Getopt instance with the given input data.  Note
that
  * this handles "short" options only.
  *
  * @param progname The name to display as the program name when
printing errors
```

```
  * @param argv The String array passed as the command line to the
program.
  * @param optstring A String containing a description of the valid
args for this program
  */
public
Getopt(String progname, String[] argv, String optstring)
{
  this(progname, argv, optstring, null, false);
}

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

/**
  * Construct a Getopt instance with given input data that is capable
of
  * parsing long options as well as short.
  *
  * @param progname The name to display as the program name when
printing errors
  * @param argv The String array passed as the command ilne to the
program
  * @param optstring A String containing a description of the valid
short args for this program
  * @param long_options An array of LongOpt objects that describes the
valid long args for this program
  */
public
Getopt(String progname, String[] argv, String optstring,
       LongOpt[] long_options)
{
  this(progname, argv, optstring, long_options, false);
}

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

/**
  * Construct a Getopt instance with given input data that is capable
of
  * parsing long options and short options.  Contrary to what you might
  * think, the flag 'long_only' does not determine whether or not we
  * scan for only long arguments.  Instead, a value of true here allows
  * long arguments to start with a '-' instead of '--' unless there is
a
  * conflict with a short option name.
  *
  * @param progname The name to display as the program name when
printing errors
  * @param argv The String array passed as the command ilne to the
program
  * @param optstring A String containing a description of the valid
short args for this program
  * @param long_options An array of LongOpt objects that describes the
valid long args for this program
```

```
  * @param long_only true if long options that do not conflict with
short options can start with a '-' as well as '--'
  */
public
Getopt(String progname, String[] argv, String optstring,
       LongOpt[] long_options, boolean long_only)
{
  if (optstring.length() == 0)
    optstring = " ";

  // This function is essentially _getopt_initialize from GNU getopt
  this.progname = progname;
  this.argv = argv;
  this.optstring = optstring;
  this.long_options = long_options;
  this.long_only = long_only;

  // Check for property "gnu.posixly_correct" to determine whether to
  // strictly follow the POSIX standard.  This replaces the
"POSIXLY_CORRECT"
  // environment variable in the C version
  if (System.getProperty("gnu.posixly_correct", null) == null)
    posixly_correct = false;
  else
    {
      posixly_correct = true;
      _messages =
PropertyResourceBundle.getBundle("gnu/getopt/MessagesBundle",
                                                       Locale.US);
    }

  // Determine how to handle the ordering of options and non-options
  if (optstring.charAt(0) == '-')
    {
      ordering = RETURN_IN_ORDER;
      if (optstring.length() > 1)
        this.optstring = optstring.substring(1);
    }
  else if (optstring.charAt(0) == '+')
    {
      ordering = REQUIRE_ORDER;
      if (optstring.length() > 1)
        this.optstring = optstring.substring(1);
    }
  else if (posixly_correct)
    {
      ordering = REQUIRE_ORDER;
    }
  else
    {
      ordering = PERMUTE; // The normal default case
    }
}

/*************************************************************************
****/
```

```
/*
 * Instance Methods
 */

/**
  * In GNU getopt, it is possible to change the string containg valid
options
  * on the fly because it is passed as an argument to getopt() each
time.  In
  * this version we do not pass the string on every call.  In order to
allow
  * dynamic option string changing, this method is provided.
  *
  * @param optstring The new option string to use
  */
public void
setOptstring(String optstring)
{
  if (optstring.length() == 0)
    optstring = " ";

  this.optstring = optstring;
}

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

/**
  * optind it the index in ARGV of the next element to be scanned.
  * This is used for communication to and from the caller
  * and for communication between successive calls to `getopt'.
  *
  * When `getopt' returns -1, this is the index of the first of the
  * non-option elements that the caller should itself scan.
  *
  * Otherwise, `optind' communicates from one call to the next
  * how much of ARGV has been scanned so far.
  */
public int
getOptind()
{
  return(optind);
}

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

/**
  * This method allows the optind index to be set manually.  Normally
this
  * is not necessary (and incorrect usage of this method can lead to
serious
  * lossage), but optind is a public symbol in GNU getopt, so this
method
  * was added to allow it to be modified by the caller if desired.
  *
  * @param optind The new value of optind
```

```
  */
public void
setOptind(int optind)
{
  this.optind = optind;
}

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

/**
  * Since in GNU getopt() the argument vector is passed back in to the
  * function every time, the caller can swap out argv on the fly.
Since
  * passing argv is not required in the Java version, this method
allows
  * the user to override argv.  Note that incorrect use of this method
can
  * lead to serious lossage.
  *
  * @param argv New argument list
  */
public void
setArgv(String[] argv)
{
  this.argv = argv;
}

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

/**
  * For communication from `getopt' to the caller.
  * When `getopt' finds an option that takes an argument,
  * the argument value is returned here.
  * Also, when `ordering' is RETURN_IN_ORDER,
  * each non-option ARGV-element is returned here.
  * No set method is provided because setting this variable has no
effect.
  */
public String
getOptarg()
{
  return(optarg);
}

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

/**
  * Normally Getopt will print a message to the standard error when an
  * invalid option is encountered.  This can be suppressed (or re-
enabled)
  * by calling this method.  There is no get method for this variable
  * because if you can't remember the state you set this to, why should
I?
  */
```

```
public void
setOpterr(boolean opterr)
{
  this.opterr = opterr;
}

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

/**
  * When getopt() encounters an invalid option, it stores the value of
that
  * option in optopt which can be retrieved with this method.  There is
  * no corresponding set method because setting this variable has no
effect.
  */
public int
getOptopt()
{
  return(optopt);
}

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

/**
  * Returns the index into the array of long options (NOT argv)
representing
  * the long option that was found.
  */
public int
getLongind()
{
  return(longind);
}

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

/**
  * Exchange the shorter segment with the far end of the longer
segment.
  * That puts the shorter segment into the right place.
  * It leaves the longer segment in the right place overall,
  * but it consists of two parts that need to be swapped next.
  * This method is used by getopt() for argument permutation.
  */
protected void
exchange(String[] argv)
{
  int bottom = first_nonopt;
  int middle = last_nonopt;
  int top = optind;
  String tem;

  while (top > middle && middle > bottom)
    {
```

```
      if (top - middle > middle - bottom)
        {
          // Bottom segment is the short one.
          int len = middle - bottom;
          int i;

          // Swap it with the top part of the top segment.
          for (i = 0; i < len; i++)
            {
              tem = argv[bottom + i];
              argv[bottom + i] = argv[top - (middle - bottom) + i];
              argv[top - (middle - bottom) + i] = tem;
            }
          // Exclude the moved bottom segment from further swapping.
          top -= len;
        }
      else
        {
          // Top segment is the short one.
          int len = top - middle;
          int i;

          // Swap it with the bottom part of the bottom segment.
          for (i = 0; i < len; i++)
            {
              tem = argv[bottom + i];
              argv[bottom + i] = argv[middle + i];
              argv[middle + i] = tem;
            }
          // Exclude the moved top segment from further swapping.
          bottom += len;
        }
    }

  // Update records for the slots the non-options now occupy.

  first_nonopt += (optind - last_nonopt);
  last_nonopt = optind;
}

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

/**
  * Check to see if an option is a valid long option.  Called by
getopt().
  * Put in a separate method because this needs to be done twice.  (The
  * C getopt authors just copy-pasted the code!).
  *
  * @param longind A buffer in which to store the 'val' field of found
LongOpt
  *
  * @return Various things depending on circumstances
  */
protected int
checkLongOption()
{
```

```
  LongOpt pfound = null;
  int nameend;
  boolean ambig;
  boolean exact;

  longopt_handled = true;
  ambig = false;
  exact = false;
  longind = -1;

  nameend = nextchar.indexOf("=");
  if (nameend == -1)
    nameend = nextchar.length();

  // Test all lnog options for either exact match or abbreviated
matches
  for (int i = 0; i < long_options.length; i++)
    {
      if (long_options[i].getName().startsWith(nextchar.substring(0,
nameend)))
        {
          if (long_options[i].getName().equals(nextchar.substring(0,
nameend)))
            {
              // Exact match found
              pfound = long_options[i];
              longind = i;
              exact = true;
              break;
            }
          else if (pfound == null)
            {
              // First nonexact match found
              pfound = long_options[i];
              longind = i;
            }
          else
            {
              // Second or later nonexact match found
              ambig = true;
            }
        }
    } // for

  // Print out an error if the option specified was ambiguous
  if (ambig && !exact)
    {
      if (opterr)
        {
          Object[] msgArgs = { progname, argv[optind] };
          System.err.println(MessageFormat.format(
                              _messages.getString("getopt.ambigious"),
                              msgArgs));
        }

       nextchar = "";
       optopt = 0;
```

```
        ++optind;

        return('?');
      }

    if (pfound != null)
      {
        ++optind;

        if (nameend != nextchar.length())
          {
            if (pfound.has_arg != LongOpt.NO_ARGUMENT)
              {
                if (nextchar.substring(nameend).length() > 1)
                  optarg = nextchar.substring(nameend+1);
                else
                  optarg = "";
              }
            else
              {
                if (opterr)
                  {
                    // -- option
                    if (argv[optind - 1].startsWith("--"))
                      {
                        Object[] msgArgs = { progname, pfound.name };
                        System.err.println(MessageFormat.format(

_messages.getString("getopt.arguments1"),
                                    msgArgs));
                      }
                    // +option or -option
                    else
                      {
                        Object[] msgArgs = { progname, new
                                    Character(argv[optind-
1].charAt(0)).toString(),
                                    pfound.name };
                        System.err.println(MessageFormat.format(

_messages.getString("getopt.arguments2"),
                                    msgArgs));
                      }
                   }

                nextchar = "";
                optopt = pfound.val;

                return('?');
              }
          } // if (nameend)
        else if (pfound.has_arg == LongOpt.REQUIRED_ARGUMENT)
          {
            if (optind < argv.length)
              {
                 optarg = argv[optind];
                 ++optind;
```

```
          }
        else
          {
            if (opterr)
              {
                Object[] msgArgs = { progname, argv[optind-1] };
                System.err.println(MessageFormat.format(

_messages.getString("getopt.requires"),
                                        msgArgs));
              }

            nextchar = "";
            optopt = pfound.val;
            if (optstring.charAt(0) == ':')
              return(':');
            else
              return('?');
          }
      } // else if (pfound)

    nextchar = "";

    if (pfound.flag != null)
      {
        pfound.flag.setLength(0);
        pfound.flag.append(pfound.val);

        return(0);
      }

    return(pfound.val);
  } // if (pfound != null)

  longopt_handled = false;

  return(0);
}

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

/**
  * This method returns a char that is the current option that has been
  * parsed from the command line.  If the option takes an argument,
then
  * the internal variable 'optarg' is set which is a String
representing
  * the the value of the argument.  This value can be retrieved by the
  * caller using the getOptarg() method.  If an invalid option is
found,
  * an error message is printed and a '?' is returned.  The name of the
  * invalid option character can be retrieved by calling the
getOptopt()
  * method.  When there are no more options to be scanned, this method
  * returns -1.  The index of first non-option element in argv can be
  * retrieved with the getOptind() method.
```

```
  *
  * @return Various things as described above
  */
public int
getopt()
{
  optarg = null;

  if (endparse == true)
    return(-1);

  if ((nextchar == null) || (nextchar.equals("")))
    {
      // If we have just processed some options following some non-
options,
      //  exchange them so that the options come first.
      if (last_nonopt > optind)
        last_nonopt = optind;
      if (first_nonopt > optind)
        first_nonopt = optind;

      if (ordering == PERMUTE)
        {
          // If we have just processed some options following some non-
options,
          // exchange them so that the options come first.
          if ((first_nonopt != last_nonopt) && (last_nonopt != optind))
            exchange(argv);
          else if (last_nonopt != optind)
            first_nonopt = optind;

          // Skip any additional non-options
          // and extend the range of non-options previously skipped.
          while ((optind < argv.length) && (argv[optind].equals("") ||
            (argv[optind].charAt(0) != '-') || argv[optind].equals("-
")))
            {
              optind++;
            }

          last_nonopt = optind;
        }

      // The special ARGV-element `--' means premature end of options.
      // Skip it like a null option,
      // then exchange with previous non-options as if it were an
option,
      // then skip everything else like a non-option.
      if ((optind != argv.length) && argv[optind].equals("--"))
        {
          optind++;

          if ((first_nonopt != last_nonopt) && (last_nonopt != optind))
            exchange (argv);
          else if (first_nonopt == last_nonopt)
            first_nonopt = optind;
```

```
          last_nonopt = argv.length;

          optind = argv.length;
        }

      // If we have done all the ARGV-elements, stop the scan
      // and back over any non-options that we skipped and permuted.
      if (optind == argv.length)
        {
          // Set the next-arg-index to point at the non-options
          // that we previously skipped, so the caller will digest
them.
          if (first_nonopt != last_nonopt)
            optind = first_nonopt;

          return(-1);
        }

      // If we have come to a non-option and did not permute it,
      // either stop the scan or describe it to the caller and pass it
by.
      if (argv[optind].equals("") || (argv[optind].charAt(0) != '-') ||
          argv[optind].equals("-"))
        {
          if (ordering == REQUIRE_ORDER)
            return(-1);

          optarg = argv[optind++];
          return(1);
        }

      // We have found another option-ARGV-element.
      // Skip the initial punctuation.
      if (argv[optind].startsWith("--"))
        nextchar = argv[optind].substring(2);
      else
        nextchar = argv[optind].substring(1);
   }

  // Decode the current option-ARGV-element.

  /* Check whether the ARGV-element is a long option.

     If long_only and the ARGV-element has the form "-f", where f is
     a valid short option, don't consider it an abbreviated form of
     a long option that starts with f.  Otherwise there would be no
     way to give the -f short option.

     On the other hand, if there's a long option "fubar" and
     the ARGV-element is "-fu", do consider that an abbreviation of
     the long option, just like "--fu", and not "-f" with arg "u".

     This distinction seems to be the most useful approach.  */
  if ((long_options != null) && (argv[optind].startsWith("--")
      || (long_only && ((argv[optind].length()  > 2) ||
      (optstring.indexOf(argv[optind].charAt(1)) == -1)))))
    {
```

```
       int c = checkLongOption();

       if (longopt_handled)
         return(c);

      // Can't find it as a long option.  If this is not
getopt_long_only,
      // or the option starts with '--' or is not a valid short
      // option, then it's an error.
      // Otherwise interpret it as a short option.
      if (!long_only || argv[optind].startsWith("--")
        || (optstring.indexOf(nextchar.charAt(0)) == -1))
        {
          if (opterr)
            {
              if (argv[optind].startsWith("--"))
                {
                  Object[] msgArgs = { progname, nextchar };
                  System.err.println(MessageFormat.format(

_messages.getString("getopt.unrecognized"),
                                      msgArgs));
                }
              else
                {
                  Object[] msgArgs = { progname, new

Character(argv[optind].charAt(0)).toString(),
                                      nextchar };
                  System.err.println(MessageFormat.format(

_messages.getString("getopt.unrecognized2"),
                                      msgArgs));
                }
            }

          nextchar = "";
          ++optind;
          optopt = 0;

          return('?');
        }
    } // if (longopts)

  // Look at and handle the next short option-character */
  int c = nextchar.charAt(0); //**** Do we need to check for empty str?
  if (nextchar.length() > 1)
    nextchar = nextchar.substring(1);
  else
    nextchar = "";

  String temp = null;
  if (optstring.indexOf(c) != -1)
    temp = optstring.substring(optstring.indexOf(c));

  if (nextchar.equals(""))
    ++optind;
```

```
  if ((temp == null) || (c == ':'))
    {
      if (opterr)
        {
          if (posixly_correct)
            {
              // 1003.2 specifies the format of this message
              Object[] msgArgs = { progname, new
                                   Character((char)c).toString() };
              System.err.println(MessageFormat.format(
                           _messages.getString("getopt.illegal"),
msgArgs));
            }
          else
            {
              Object[] msgArgs = { progname, new
                                   Character((char)c).toString() };
              System.err.println(MessageFormat.format(
                           _messages.getString("getopt.invalid"),
msgArgs));
            }
        }

      optopt = c;

      return('?');
    }

  // Convenience. Treat POSIX -W foo same as long option --foo
  if ((temp.charAt(0) == 'W') && (temp.length() > 1) && (temp.charAt(1)
== ';'))
    {
      if (!nextchar.equals(""))
        {
          optarg = nextchar;
        }
      // No further cars in this argv element and no more argv elements
      else if (optind == argv.length)
        {
          if (opterr)
            {
              // 1003.2 specifies the format of this message.
              Object[] msgArgs = { progname, new
                                   Character((char)c).toString() };
              System.err.println(MessageFormat.format(
                           _messages.getString("getopt.requires2"),
msgArgs));
            }

          optopt = c;
          if (optstring.charAt(0) == ':')
            return(':');
          else
            return('?');
        }
      else
```

```
        {
          // We already incremented `optind' once;
          // increment it again when taking next ARGV-elt as argument.
          nextchar = argv[optind];
          optarg  = argv[optind];
        }

      c = checkLongOption();

      if (longopt_handled)
        return(c);
      else
        // Let the application handle it
        {
          nextchar = null;
          ++optind;
          return('W');
        }
    }

  if ((temp.length() > 1) && (temp.charAt(1) == ':'))
    {
      if ((temp.length() > 2) && (temp.charAt(2) == ':'))
        // This is an option that accepts and argument optionally
        {
          if (!nextchar.equals(""))
            {
               optarg = nextchar;
               ++optind;
            }
          else
            {
              optarg = null;
            }

          nextchar = null;
        }
      else
        {
          if (!nextchar.equals(""))
            {
              optarg = nextchar;
              ++optind;
            }
          else if (optind == argv.length)
            {
              if (opterr)
                {
                  // 1003.2 specifies the format of this message
                  Object[] msgArgs = { progname, new
                                       Character((char)c).toString() };
                  System.err.println(MessageFormat.format(
                            _messages.getString("getopt.requires2"),
msgArgs));
                }

              optopt = c;
```

```
              if (optstring.charAt(0) == ':')
                return(':');
              else
                return('?');
            }
          else
            {
              optarg = argv[optind];
              ++optind;

              // Ok, here's an obscure Posix case.  If we have o:, and
              // we get -o -- foo, then we're supposed to skip the --,
              // end parsing of options, and make foo an operand to -o.
              // Only do this in Posix mode.
              if ((posixly_correct) && optarg.equals("--"))
                {
                  // If end of argv, error out
                  if (optind == argv.length)
                    {
                      if (opterr)
                        {
                          // 1003.2 specifies the format of this
message
                          Object[] msgArgs = { progname, new

Character((char)c).toString() };
                          System.err.println(MessageFormat.format(
                             _messages.getString("getopt.requires2"),
msgArgs));
                        }

                      optopt = c;

                      if (optstring.charAt(0) == ':')
                        return(':');
                      else
                        return('?');
                    }

                  // Set new optarg and set to end
                  // Don't permute as we do on -- up above since we
                  // know we aren't in permute mode because of Posix.
                  optarg = argv[optind];
                  ++optind;
                  first_nonopt = optind;
                  last_nonopt = argv.length;
                  endparse = true;
                }
            }

          nextchar = null;
        }
    }

  return(c);
}
```

```
} // Class Getopt


/**************************************************************************
****
/* LongOpt.java -- Long option object for Getopt
/*
/* Copyright (c) 1998 by Aaron M. Renn (arenn@urbanophile.com)
/*
/* This program is free software; you can redistribute it and/or modify
/* it under the terms of the GNU Library General Public License as
published
/* by  the Free Software Foundation; either version 2 of the License or
/* (at your option) any later version.
/*
/* This program is distributed in the hope that it will be useful, but
/* WITHOUT ANY WARRANTY; without even the implied warranty of
/* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
/* GNU Library General Public License for more details.
/*
/* You should have received a copy of the GNU Library General Public
License
/* along with this program; see the file COPYING.LIB.  If not, write to
/* the Free Software Foundation Inc., 59 Temple Place - Suite 330,
/* Boston, MA  02111-1307 USA
/**************************************************************************
****/

//package gnu.getopt;

import java.util.Locale;
import java.util.ResourceBundle;
import java.util.PropertyResourceBundle;
import java.text.MessageFormat;

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

/**
  * This object represents the definition of a long option in the Java
port
  * of GNU getopt.  An array of LongOpt objects is passed to the Getopt
  * object to define the list of valid long options for a given parsing
  * session.  Refer to the getopt documentation for details on the
  * format of long options.
  *
  * @version 1.0.5
  * @author Aaron M. Renn (arenn@urbanophile.com)
  *
  * @see Getopt
  */
public class LongOpt extends Object
{

/**************************************************************************
****/
```

```
/*
 * Class Variables
 */

/**
  * Constant value used for the "has_arg" constructor argument.  This
  * value indicates that the option takes no argument.
  */
public static final int NO_ARGUMENT = 0;

/**
  * Constant value used for the "has_arg" constructor argument.  This
  * value indicates that the option takes an argument that is required.
  */
public static final int REQUIRED_ARGUMENT = 1;

/**
  * Constant value used for the "has_arg" constructor argument.  This
  * value indicates that the option takes an argument that is optional.
  */
public static final int OPTIONAL_ARGUMENT = 2;

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

/*
 * Instance Variables
 */

/**
  * The name of the long option
  */
protected String name;

/**
  * Indicates whether the option has no argument, a required argument,
or
  * an optional argument.
  */
protected int has_arg;

/**
  * If this variable is not null, then the value stored in "val" is
stored
  * here when this long option is encountered.  If this is null, the
value
  * stored in "val" is treated as the name of an equivalent short
option.
  */
protected StringBuffer flag;

/**
  * The value to store in "flag" if flag is not null, otherwise the
  * equivalent short option character for this long option.
  */
protected int val;
```

```
/**
  * Localized strings for error messages
  */
private ResourceBundle _messages = PropertyResourceBundle.getBundle(
                              "gnu/getopt/MessagesBundle",
Locale.getDefault());

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

/*
 * Constructors
 */

/**
  * Create a new LongOpt object with the given parameter values.  If
the
  * value passed as has_arg is not valid, then an exception is thrown.
  *
  * @param name The long option String.
  * @param has_arg Indicates whether the option has no argument
(NO_ARGUMENT), a required argument (REQUIRED_ARGUMENT) or an optional
argument (OPTIONAL_ARGUMENT).
  * @param flag If non-null, this is a location to store the value of
"val" when this option is encountered, otherwise "val" is treated as
the equivalent short option character.
  * @param val The value to return for this long option, or the
equivalent single letter option to emulate if flag is null.
  *
  * @exception IllegalArgumentException If the has_arg param is not one
of NO_ARGUMENT, REQUIRED_ARGUMENT or OPTIONAL_ARGUMENT.
  */
public
LongOpt(String name, int has_arg,
        StringBuffer flag, int val) throws IllegalArgumentException
{
  // Validate has_arg
  if ((has_arg != NO_ARGUMENT) && (has_arg != REQUIRED_ARGUMENT)
     && (has_arg != OPTIONAL_ARGUMENT))
    {
      Object[] msgArgs = { new Integer(has_arg).toString() };
      throw new IllegalArgumentException(MessageFormat.format(
                    _messages.getString("getopt.invalidValue"),
msgArgs));
    }

  // Store off values
  this.name = name;
  this.has_arg = has_arg;
  this.flag = flag;
  this.val = val;
}

/**********************************************************************
****/
```

```
/**
  * Returns the name of this LongOpt as a String
  *
  * @return Then name of the long option
  */
public String
getName()
{
  return(name);
}

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

/**
  * Returns the value set for the 'has_arg' field for this long option
  *
  * @return The value of 'has_arg'
  */
public int
getHasArg()
{
  return(has_arg);
}

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

/**
  * Returns the value of the 'flag' field for this long option
  *
  * @return The value of 'flag'
  */
public StringBuffer
getFlag()
{
  return(flag);
}

/**
  * Returns the value of the 'val' field for this long option
  *
  * @return The value of 'val'
  */
public int
getVal()
{
  return(val);
}

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

} // Class LongOpt

/*
 *      MixingAudioInputStream.java
```

```
 *
 *      This file is part of the Java Sound Examples.
 *
 *      This code follows an idea of Paul Sorenson.
 */

/*
 *  Copyright (c) 1999 - 2001 by Matthias Pfisterer
<Matthias.Pfisterer@web.de>
 *
 *
 *   This program is free software; you can redistribute it and/or
modify
 *   it under the terms of the GNU Library General Public License as
published
 *   by the Free Software Foundation; either version 2 of the License,
or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *   GNU Library General Public License for more details.
 *
 *   You should have received a copy of the GNU Library General Public
 *   License along with this program; if not, write to the Free
Software
 *   Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 *
 */



import  java.io.ByteArrayInputStream;
import  java.io.InputStream;
import  java.io.IOException;
import  java.net.URL;

import  java.util.ArrayList;
import  java.util.Collection;
import  java.util.Iterator;
import  java.util.List;

import  javax.sound.sampled.AudioFormat;
import  javax.sound.sampled.AudioInputStream;
import  javax.sound.sampled.AudioSystem;
import  javax.sound.sampled.LineUnavailableException;
import  javax.sound.sampled.LineListener;
import  javax.sound.sampled.LineEvent;
import  javax.sound.sampled.Mixer;
import  javax.sound.sampled.SourceDataLine;
import  javax.sound.sampled.DataLine;
import  javax.sound.sampled.FloatControl;
import  javax.sound.sampled.BooleanControl;

/*
```

```
  This is a class of Tritonus. It's not one of the best ideas to use it
here.
  However, we really don't want to reimplement its functionality here.
*/

//import TConversionTool;



/**     Mixing of multiple AudioInputStreams to one AudioInputStream.
        This class takes a collection of AudioInputStreams and mixes
        them together. Being a subclass of AudioInputStream itself,
        reading from instances of this class behaves as if the mixdown
        result of the input streams is read.

        @author Matthias Pfisterer
 */

//import TConversionTool;

public class MixingAudioInputStream
        extends         AudioInputStream
{
        private static final boolean    DEBUG = false;

        private List                    m_audioInputStreamList;

    //TConversionTConversionTool TConversionTool = new
TConversionTConversionTool();

        public MixingAudioInputStream(AudioFormat audioFormat,
Collection audioInputStreams)
        {


                super(new ByteArrayInputStream(new byte[0]),
                        audioFormat,
                        AudioSystem.NOT_SPECIFIED);
                if (DEBUG)
                {

System.out.println("MixingAudioInputStream.<init>(): begin");
                }
                m_audioInputStreamList = new
ArrayList(audioInputStreams);
                if (DEBUG)
                {

System.out.println("MixingAudioInputStream.<init>(): stream list:");
                        for (int i = 0; i <
m_audioInputStreamList.size(); i++)
                        {
                                System.out.println("  " +
m_audioInputStreamList.get(i));
                        }
                }
                if (DEBUG)
```

```
                {

System.out.println("MixingAudioInputStream.<init>(): end");
                }
        }



        // TODO: remove
        private boolean addAudioInputStream(AudioInputStream
audioStream)
        {
                if (DEBUG)
                {

System.out.println("MixingAudioInputStream.addAudioInputStream():
called.");
                }
                // Contract.check(audioStream != null);
                if (!getFormat().matches(audioStream.getFormat()))
                {
                        if (DEBUG)
                        {

System.out.println("MixingAudioInputStream.addAudioInputStream(): audio
formats do not match, trying to convert.");
                        }
                        AudioInputStream        asold = audioStream;
                        audioStream =
AudioSystem.getAudioInputStream(getFormat(), asold);
                        if (audioStream == null)
                        {
                                System.out.println("###
MixingAudioInputStream.addAudioInputStream(): could not convert.");
                                return false;
                        }
                        if (DEBUG)
                        {
                                System.out.println(" converted");
                        }
                }
                // Contract.check(audioStream != null);
                synchronized (m_audioInputStreamList)
                {
                        m_audioInputStreamList.add(audioStream);
                        m_audioInputStreamList.notifyAll();
                }
                if (DEBUG)
                {

System.out.println("MixingAudioInputStream.addAudioInputStream():
enqueued " + audioStream);
                }
                return true;
        }
```

```
        /**
           The maximum of the frame length of the input stream is
calculated and returned.
           If at least one of the input streams has length
           <code>AudioInputStream.NOT_SPECIFIED</code>, this value is
returned.
        */
        public long getFrameLength()
        {
                long    lLengthInFrames = 0;
                Iterator        streamIterator =
m_audioInputStreamList.iterator();
                while (streamIterator.hasNext())
                {
                        AudioInputStream        stream =
(AudioInputStream) streamIterator.next();
                        long    lLength = stream.getFrameLength();
                        if (lLength == AudioSystem.NOT_SPECIFIED)
                        {
                                return AudioSystem.NOT_SPECIFIED;
                        }
                        else
                        {
                                lLengthInFrames =
Math.max(lLengthInFrames, lLength);
                        }
                }
                return lLengthInFrames;
        }



        public int read()
                throws  IOException
        {
                if (DEBUG)
                {

System.out.println("MixingAudioInputStream.read(): begin");
                }
                int     nSample = 0;
                Iterator        streamIterator =
m_audioInputStreamList.iterator();
                while (streamIterator.hasNext())
                {
                        AudioInputStream        stream =
(AudioInputStream) streamIterator.next();
                        int     nByte = stream.read();
                        if (nByte == -1)
                        {
                                /*
                                  The end of this stream has been
signaled.
                                  We remove the stream from our list.
                                */
                                streamIterator.remove();
                                continue;
```

```
                        }
                        else
                        {
                                /*
                                  what about signed/unsigned?
                                */
                                nSample += nByte;
                        }
                }
                if (DEBUG)
                {

System.out.println("MixingAudioInputStream.read(): end");
                }
                return (byte) nSample;
        }



        public int read(byte[] abData, int nOffset, int nLength)
                throws  IOException
        {
                if (DEBUG)
                {

System.out.println("MixingAudioInputStream.read(byte[], int, int):
begin");

System.out.println("MixingAudioInputStream.read(byte[], int, int):
requested length: " + nLength);
                }
                int     nChannels = getFormat().getChannels();
                int     nFrameSize = getFormat().getFrameSize();
                /*
                  This value is in bytes. Note that it is the storage
size.
                  It may be four bytes for 24 bit samples.
                */
                int     nSampleSize = nFrameSize / nChannels;
                boolean bBigEndian = getFormat().isBigEndian();
                AudioFormat.Encoding    encoding =
getFormat().getEncoding();
                if (DEBUG)
                {

System.out.println("MixingAudioInputStream.read(byte[], int, int):
channels: " + nChannels);

System.out.println("MixingAudioInputStream.read(byte[], int, int):
frame size: " + nFrameSize);

System.out.println("MixingAudioInputStream.read(byte[], int, int):
sample size (bytes, storage size): " + nSampleSize);

System.out.println("MixingAudioInputStream.read(byte[], int, int): big
endian: " + bBigEndian);
```

```
System.out.println("MixingAudioInputStream.read(byte[], int, int):
encoding: " + encoding);
                }
                byte[]  abBuffer = new byte[nFrameSize];
                int[]   anMixedSamples = new int[nChannels];
                for (int nFrameBoundry = 0; nFrameBoundry < nLength;
nFrameBoundry += nFrameSize)
                {
                        if (DEBUG)
                        {

System.out.println("MixingAudioInputStream.read(byte[], int, int):
frame boundry: " + nFrameBoundry);
                        }
                        for (int i = 0; i < nChannels; i++)
                        {
                                anMixedSamples[i] = 0;
                        }
                        Iterator        streamIterator =
m_audioInputStreamList.iterator();
                        while (streamIterator.hasNext())
                        {
                                AudioInputStream        stream =
(AudioInputStream) streamIterator.next();
                                if (DEBUG)
                                {

System.out.println("MixingAudioInputStream.read(byte[], int, int):
AudioInputStream: " + stream);
                                }
                                int     nBytesRead =
stream.read(abBuffer, 0, nFrameSize);
                                if (DEBUG)
                                {

System.out.println("MixingAudioInputStream.read(byte[], int, int):
bytes read: " + nBytesRead);
                                }
                                /*
                                  TODO: we have to handle incomplete
reads.
                                */
                                if (nBytesRead == -1)
                                {
                                        /*
                                          The end of the current stream
has been signaled.
                                          We remove it from the list of
streams.
                                        */
                                        streamIterator.remove();
                                        continue;
                                }
                                for (int nChannel = 0; nChannel <
nChannels; nChannel++)
                                {
```

```
                                        int     nBufferOffset =
nChannel * nSampleSize;
                                        int     nSampleToAdd = 0;
                                        if
(encoding.equals(AudioFormat.Encoding.PCM_SIGNED))
                                        {
                                                switch (nSampleSize)
                                                {
                                                case 1:
                                                        nSampleToAdd =
abBuffer[nBufferOffset];
                                                        break;
                                                case 2:
                                                        nSampleToAdd =
TConversionTool.bytesToInt16(abBuffer, nBufferOffset, bBigEndian);
                                                        break;
                                                case 3:
                                                        nSampleToAdd =
TConversionTool.bytesToInt24(abBuffer, nBufferOffset, bBigEndian);
                                                        break;
                                                case 4:
                                                        nSampleToAdd =
TConversionTool.bytesToInt32(abBuffer, nBufferOffset, bBigEndian);
                                                        break;
                                                }
                                        }
                                        // TODO: pcm unsigned
                                        else if
(encoding.equals(AudioFormat.Encoding.ALAW))
                                        {
                                                nSampleToAdd =
TConversionTool.alaw2linear(abBuffer[nBufferOffset]);
                                        }
                                        else if
(encoding.equals(AudioFormat.Encoding.ULAW))
                                        {
                                                nSampleToAdd =
TConversionTool.ulaw2linear(abBuffer[nBufferOffset]);
                                        }
                                        anMixedSamples[nChannel] +=
nSampleToAdd;
                                } // loop over channels
                        } // loop over streams
                        if (DEBUG)
                        {

System.out.println("MixingAudioInputStream.read(byte[], int, int):
starting to write to buffer passed by caller");
                        }
                        for (int nChannel = 0; nChannel < nChannels;
nChannel++)
                        {
                                if (DEBUG)
                                {

System.out.println("MixingAudioInputStream.read(byte[], int, int):
channel: " + nChannel);
```

```
                                }
                                int     nBufferOffset = nOffset +
nFrameBoundry /* * nFrameSize*/ + nChannel * nSampleSize;
                                if (DEBUG)
                                {

System.out.println("MixingAudioInputStream.read(byte[], int, int):
buffer offset: " + nBufferOffset);
                                }
                                if
(encoding.equals(AudioFormat.Encoding.PCM_SIGNED))
                                {
                                        switch (nSampleSize)
                                        {
                                        case 1:
                                                abData[nBufferOffset] =
(byte) anMixedSamples[nChannel];
                                                break;
                                        case 2:

TConversionTool.intToBytes16(anMixedSamples[nChannel], abData,
nBufferOffset, bBigEndian);
                                                break;
                                        case 3:

TConversionTool.intToBytes24(anMixedSamples[nChannel], abData,
nBufferOffset, bBigEndian);
                                                break;
                                        case 4:

TConversionTool.intToBytes32(anMixedSamples[nChannel], abData,
nBufferOffset, bBigEndian);
                                                break;
                                        }
                                }
                                // TODO: pcm unsigned
                                else if
(encoding.equals(AudioFormat.Encoding.ALAW))
                                {
                                        abData[nBufferOffset] =
TConversionTool.linear2alaw((short) anMixedSamples[nChannel]);
                                }
                                else if
(encoding.equals(AudioFormat.Encoding.ULAW))
                                {
                                        abData[nBufferOffset] =
TConversionTool.linear2ulaw(anMixedSamples[nChannel]);
                                }
                        } // (final) loop over channels
                } // loop over frames
                if (DEBUG)
                {

System.out.println("MixingAudioInputStream.read(byte[], int, int):
end");
                }
                // TODO: return a useful value
```

```
                return nLength;
        }



        /**
           calls skip() on all input streams. There is no way to assure
that the number of
           bytes really skipped is the same for all input streams. Due
to that, this
           method always returns the passed value. In other words: the
return value
           is useless (better ideas appreciated).
        */
        public long skip(long lLength)
                throws  IOException
        {
                int     nAvailable = 0;
                Iterator        streamIterator =
m_audioInputStreamList.iterator();
                while (streamIterator.hasNext())
                {
                        AudioInputStream        stream =
(AudioInputStream) streamIterator.next();
                        stream.skip(lLength);
                }
                return lLength;
        }



        /**
           The minimum of available() of all input stream is calculated
and returned.
        */
        public int available()
                throws  IOException
        {
                int     nAvailable = 0;
                Iterator        streamIterator =
m_audioInputStreamList.iterator();
                while (streamIterator.hasNext())
                {
                        AudioInputStream        stream =
(AudioInputStream) streamIterator.next();
                        nAvailable = Math.min(nAvailable,
stream.available());
                }
                return nAvailable;
        }



        public void close()
                throws  IOException
        {
                // TODO: should we close all streams in the list?
```

```
        }


        /**
           Calls mark() on all input streams.
        */
        public void mark(int nReadLimit)
        {
                Iterator        streamIterator =
m_audioInputStreamList.iterator();
                while (streamIterator.hasNext())
                {
                        AudioInputStream        stream =
(AudioInputStream) streamIterator.next();
                        stream.mark(nReadLimit);
                }
        }


        /**
           Calls reset() on all input streams.
        */
        public void reset()
                throws  IOException
        {
                Iterator        streamIterator =
m_audioInputStreamList.iterator();
                while (streamIterator.hasNext())
                {
                        AudioInputStream        stream =
(AudioInputStream) streamIterator.next();
                        stream.reset();
                }
        }



        /**
           returns true if all input stream return true for
markSupported().
        */
        public boolean markSupported()
        {
                Iterator        streamIterator =
m_audioInputStreamList.iterator();
                while (streamIterator.hasNext())
                {
                        AudioInputStream        stream =
(AudioInputStream) streamIterator.next();
                        if (! stream.markSupported())
                        {
                                return false;
                        }
                }
                return true;
        }
```

```
}


/*** MixingAudioInputStream.java ***/
/*
 *      SequenceAudioInputStream.java
 *
 *      This file is part of the Java Sound Examples.
 */

/*
 *  Copyright (c) 1999 - 2001 by Matthias Pfisterer
<Matthias.Pfisterer@web.de>
 *
 *
 *   This program is free software; you can redistribute it and/or
modify
 *   it under the terms of the GNU Library General Public License as
published
 *   by the Free Software Foundation; either version 2 of the License,
or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *   GNU Library General Public License for more details.
 *
 *   You should have received a copy of the GNU Library General Public
 *   License along with this program; if not, write to the Free
Software
 *   Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 *
 */



import  java.io.ByteArrayInputStream;
import  java.io.InputStream;
import  java.io.IOException;
import  java.net.URL;

import  java.util.ArrayList;
import  java.util.Collection;
import  java.util.Iterator;
import  java.util.List;

import  javax.sound.sampled.AudioFormat;
import  javax.sound.sampled.AudioInputStream;
```

```
import  javax.sound.sampled.AudioSystem;
import  javax.sound.sampled.LineUnavailableException;
import  javax.sound.sampled.LineListener;
import  javax.sound.sampled.LineEvent;
import  javax.sound.sampled.Mixer;
import  javax.sound.sampled.SourceDataLine;
import  javax.sound.sampled.DataLine;
import  javax.sound.sampled.FloatControl;
import  javax.sound.sampled.BooleanControl;



public class SequenceAudioInputStream
        extends         AudioInputStream
{
        private static final boolean    DEBUG = true;

        private List                    m_audioInputStreamList;
        private int                     m_nCurrentStream;



        public SequenceAudioInputStream(AudioFormat audioFormat,
Collection audioInputStreams)
        {
                super(new ByteArrayInputStream(new byte[0]),
audioFormat, AudioSystem.NOT_SPECIFIED);
                m_audioInputStreamList = new
ArrayList(audioInputStreams);
                m_nCurrentStream = 0;
        }



        // TODO: remove
        private boolean addAudioInputStream(AudioInputStream
audioStream)
        {
                if (DEBUG)
                {

System.out.println("SequenceAudioInputStream.addAudioInputStream():
called.");
                }
                // Contract.check(audioStream != null);
                if (!getFormat().matches(audioStream.getFormat()))
                {
                        if (DEBUG)
                        {

System.out.println("SequenceAudioInputStream.addAudioInputStream():
audio formats do not match, trying to convert.");
                        }
                        AudioInputStream        asold = audioStream;
                        audioStream =
AudioSystem.getAudioInputStream(getFormat(), asold);
                        if (audioStream == null)
```

```
                        {
                                System.out.println("###
SequenceAudioInputStream.addAudioInputStream(): could not convert.");
                                return false;
                        }
                        if (DEBUG)
                        {
                                System.out.println(" converted");
                        }
                }
                // Contract.check(audioStream != null);
                synchronized (m_audioInputStreamList)
                {
                        m_audioInputStreamList.add(audioStream);
                        m_audioInputStreamList.notifyAll();
                }
                if (DEBUG)
                {

System.out.println("SequenceAudioInputStream.addAudioInputStream():
enqueued " + audioStream);
                }
                return true;
        }



        private AudioInputStream getCurrentStream()
        {
                return (AudioInputStream)
m_audioInputStreamList.get(m_nCurrentStream);
        }



        private boolean advanceStream()
        {
                m_nCurrentStream++;
                boolean bAnotherStreamAvailable = (m_nCurrentStream <
m_audioInputStreamList.size());
                return bAnotherStreamAvailable;
        }



        public long getFrameLength()
        {
                long    lLengthInFrames = 0;
                Iterator        streamIterator =
m_audioInputStreamList.iterator();
                while (streamIterator.hasNext())
                {
                        AudioInputStream        stream =
(AudioInputStream) streamIterator.next();
                        long    lLength = stream.getFrameLength();
                        if (lLength == AudioSystem.NOT_SPECIFIED)
                        {
```

```
                        return AudioSystem.NOT_SPECIFIED;
                    }
                    else
                    {
                        lLengthInFrames += lLength;
                    }
                }
                return lLengthInFrames;
        }



        public int read()
                throws  IOException
        {
                AudioInputStream        stream = getCurrentStream();
                int     nByte = stream.read();
                if (nByte == -1)
                {
                    /*
                      The end of the current stream has been
signaled.
                      We try to advance to the next stream.
                    */
                    boolean bAnotherStreamAvailable =
advanceStream();
                    if (bAnotherStreamAvailable)
                    {
                        /*
                          There is another stream. We recurse
into this method
                          to read from it.
                        */
                        return read();
                    }
                    else
                    {
                        /*
                          No more data. We signal EOF.
                        */
                        return -1;
                    }
                }
                else
                {
                    /*
                      The most common case: We return the byte.
                    */
                    return nByte;
                }
        }



        public int read(byte[] abData, int nOffset, int nLength)
                throws  IOException
        {
```

```
                AudioInputStream        stream = getCurrentStream();
                int     nBytesRead = stream.read(abData, nOffset,
nLength);
                if (nBytesRead == -1)
                {
                        /*
                          The end of the current stream has been
signaled.
                          We try to advance to the next stream.
                        */
                        boolean bAnotherStreamAvailable =
advanceStream();
                        if (bAnotherStreamAvailable)
                        {
                                /*
                                  There is another stream. We recurse
into this method
                                  to read from it.
                                */
                                return read(abData, nOffset, nLength);
                        }
                        else
                        {
                                /*
                                  No more data. We signal EOF.
                                */
                                return -1;
                        }
                }
                else
                {
                        /*
                          The most common case: We return the length.
                        */
                        return nBytesRead;
                }
        }



        public long skip(long lLength)
                throws  IOException
        {
                throw new IOException("skip() is not implemented in
class SequenceInputStream. Mail <Matthias.Pfisterer@web.de> if you need
this feature.");
        }



        public int available()
                throws  IOException
        {
                return getCurrentStream().available();
        }
```

```
        public void close()
                throws  IOException
        {
                // TODO: should we close all streams in the list?
        }



        public void mark(int nReadLimit)
        {
                throw new RuntimeException("mark() is not implemented
in class SequenceInputStream. Mail <Matthias.Pfisterer@web.de> if you
need this feature.");
        }



        public void reset()
                throws  IOException
        {
                throw new IOException("reset() is not implemented in
class SequenceInputStream. Mail <Matthias.Pfisterer@web.de> if you need
this feature.");
        }



        public boolean markSupported()
        {
                return false;
        }
}



/*** SequenceAudioInputStream.java ***/
/*
 *    TConversionTool.java
 */

/*
 *  Copyright (c) 1999,2000 by Florian Bomers <florian@bome.com>
 *  Copyright (c) 2000 by Matthias Pfisterer
<matthias.pfisterer@gmx.de>
 *
 *
 *   This program is free software; you can redistribute it and/or
modify
 *   it under the terms of the GNU Library General Public License as
published
 *   by the Free Software Foundation; either version 2 of the License,
or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *   GNU Library General Public License for more details.
 *
 *   You should have received a copy of the GNU Library General Public
 *   License along with this program; if not, write to the Free
Software
 *   Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 *
 */




/**
 * Useful methods for converting audio data.
 *
 * @author Florian Bomers
 * @author Matthias Pfisterer
 */

/*
For convenience, a list of available methods is maintained here.
Some hints:
- buffers: always byte arrays
- offsets: always in bytes
- sampleCount: number of SAMPLES to read/write, as opposed to FRAMES or
BYTES!
- when in buffer and out buffer are given, the data is copied,
  otherwise it is replaced in the same buffer (buffer size is not
checked!)
- a number (except "2") gives the number of bits in which format
  the samples have to be.
- >8 bits per sample is always treated signed.
- all functions are tried to be optimized - hints welcome !


** "high level" methods **
changeOrderOrSign(buffer, nOffset, nByteLength, nBytesPerSample)
changeOrderOrSign(inBuffer, nInOffset, outBuffer, nOutOffset,
nByteLength, nBytesPerSample)


** PCM byte order and sign conversion **
void  convertSign8(buffer, byteOffset, sampleCount)
void  swapOrder16(buffer, byteOffset, sampleCount)
void  swapOrder24(buffer, byteOffset, sampleCount)
void  swapOrder32(buffer, byteOffset, sampleCount)
void  convertSign8(inBuffer, inByteOffset, outBuffer, outByteOffset,
sampleCount)
void  swapOrder16(inBuffer, inByteOffset, outBuffer, outByteOffset,
sampleCount)
void  swapOrder24(inBuffer, inByteOffset, outBuffer, outByteOffset,
sampleCount)
void  swapOrder32(inBuffer, inByteOffset, outBuffer, outByteOffset,
sampleCount)
```

```
** conversion functions for byte arrays **
** these are for reference to see how to implement these conversions **
short       bytesToShort16(highByte, lowByte)
short       bytesToShort16(buffer, byteOffset, bigEndian)
short       bytesToInt16(highByte, lowByte)
short       bytesToInt16(buffer, byteOffset, bigEndian)
short       bytesToInt24(buffer, byteOffset, bigEndian)
short       bytesToInt32(buffer, byteOffset, bigEndian)
void  shortToBytes16(sample, buffer, byteOffset, bigEndian)
void  intToBytes24(sample, buffer, byteOffset, bigEndian)
void  intToBytes32(sample, buffer, byteOffset, bigEndian)


** ULAW <-> PCM **
byte  linear2ulaw(int sample)
short       ulaw2linear(int ulawbyte)
void  pcm162ulaw(buffer, byteOffset, sampleCount, bigEndian)
void  pcm162ulaw(inBuffer, inByteOffset, outBuffer, outByteOffset,
sampleCount, bigEndian)
void  pcm82ulaw(buffer, byteOffset, sampleCount, signed)
void  pcm82ulaw(inBuffer, inByteOffset, outBuffer, outByteOffset,
sampleCount, signed)
void  ulaw2pcm16(inBuffer, inByteOffset, outBuffer, outByteOffset,
sampleCount, bigEndian)
void  ulaw2pcm8(buffer, byteOffset, sampleCount, signed)
void  ulaw2pcm8(inBuffer, inByteOffset, outBuffer, outByteOffset,
sampleCount, signed)


** ALAW <-> PCM **
byte linear2alaw(short pcm_val)
short alaw2linear(byte ulawbyte)
void pcm162alaw(inBuffer, inByteOffset, outBuffer, outByteOffset,
sampleCount, bigEndian)
void pcm162alaw(buffer, byteOffset, sampleCount, bigEndian)
void pcm82alaw(buffer, byteOffset, sampleCount, signed)
void pcm82alaw(inBuffer, inByteOffset, outBuffer, outByteOffset,
sampleCount, signed)
void alaw2pcm16(inBuffer, inByteOffset, outBuffer, outByteOffset,
sampleCount, bigEndian)
void alaw2pcm8(buffer, byteOffset, sampleCount, signed)
void alaw2pcm8(inBuffer, inByteOffset, outBuffer, outByteOffset,
sampleCount, signed)


** ULAW <-> ALAW **
byte  ulaw2alaw(byte sample)
void  ulaw2alaw(buffer, byteOffset, sampleCount)
void  ulaw2alaw(inBuffer, inByteOffset, outBuffer, outByteOffset,
sampleCount)
byte  alaw2ulaw(byte sample)
void  alaw2ulaw(buffer, byteOffset, sampleCount)
void  alaw2ulaw(inBuffer, inByteOffset, outBuffer, outByteOffset,
sampleCount)

*/
```

```
public class TConversionTool {

      ///////////////// sign/byte-order conversion
////////////////////////////////

    public TConversionTool(){

    }

      public static void convertSign8(byte[] buffer, int byteOffset,
int sampleCount) {
            sampleCount+=byteOffset;
            for (int i=byteOffset; i<sampleCount; i++) {
                  buffer[i]+=128;
            }
      }

      public static void swapOrder16(byte[] buffer, int byteOffset, int
sampleCount) {
            int byteMax=sampleCount*2+byteOffset-1;
            int i=byteOffset;
            while (i<byteMax) {
                  byte h=buffer[i];
                  buffer[i]=buffer[++i];
                  buffer[i++]=h;
            }
      }

      public static void swapOrder24(byte[] buffer, int byteOffset, int
sampleCount) {
            int byteMax=sampleCount*3+byteOffset-2;
            int i=byteOffset;
            while (i<byteMax) {
                  byte h=buffer[i];
                  buffer[i]=buffer[++i+1];
                  buffer[++i]=h;
                  i++;
            }
      }

      public static void swapOrder32(byte[] buffer, int byteOffset, int
sampleCount) {
            int byteMax=sampleCount*4+byteOffset-3;
            int i=byteOffset;
            while (i<byteMax) {
                  byte h=buffer[i];
                  buffer[i]=buffer[i+3];
                  buffer[i+3]=h;
                  i++;
                  h=buffer[i];
                  buffer[i]=buffer[++i];
                  buffer[i++]=h;
                  i++;
            }
      }
```

```
      public static void convertSign8(byte[] inBuffer, int
inByteOffset,
                                      byte[] outBuffer, int
outByteOffset, int sampleCount) {
            while (sampleCount>0) {

      outBuffer[outByteOffset++]=(byte)(inBuffer[inByteOffset++]+128);
                  sampleCount--;
            }
      }

      public static void swapOrder16(byte[] inBuffer, int inByteOffset,
                                      byte[] outBuffer, int
outByteOffset, int sampleCount) {
            while (sampleCount>0) {
                  outBuffer[outByteOffset++]=inBuffer[inByteOffset+1];
                  outBuffer[outByteOffset++]=inBuffer[inByteOffset++];
                  inByteOffset++;
                  sampleCount--;
            }
      }

      public static void swapOrder24(byte[] inBuffer, int inByteOffset,
                                      byte[] outBuffer, int
outByteOffset, int sampleCount) {
            while (sampleCount>0) {
                  outBuffer[outByteOffset++]=inBuffer[inByteOffset+2];
                  outByteOffset++;
                  outBuffer[outByteOffset++]=inBuffer[inByteOffset++];
                  inByteOffset++;
                  inByteOffset++;
                  sampleCount--;
            }
      }

      public static void swapOrder32(byte[] inBuffer, int inByteOffset,
                                      byte[] outBuffer, int
outByteOffset, int sampleCount) {
            while (sampleCount>0) {
                  outBuffer[outByteOffset++]=inBuffer[inByteOffset+3];
                  outBuffer[outByteOffset++]=inBuffer[inByteOffset+2];
                  outBuffer[outByteOffset++]=inBuffer[inByteOffset+1];
                  outBuffer[outByteOffset++]=inBuffer[inByteOffset++];
                  inByteOffset++;
                  inByteOffset++;
                  inByteOffset++;
                  sampleCount--;
            }
      }


      ///////////////// conversion functions for byte arrays
//////////////////////////


      /**
```

```
       * Converts 2 bytes to a signed sample of type
<code>short</code>.
       * <p> This is a reference function.
       */
      public static short bytesToShort16(byte highByte, byte lowByte) {
            return (short) ((highByte<<8) | (lowByte & 0xFF));
      }

      /**
       * Converts 2 successive bytes starting at
<code>byteOffset</code> in
       * <code>buffer</code> to a signed sample of type
<code>short</code>.
       * <p>
       * For little endian, buffer[byteOffset] is interpreted as low
byte,
       * whereas it is interpreted as high byte in big endian.
       * <p> This is a reference function.
       */
      public static short bytesToShort16(byte[] buffer, int byteOffset,
boolean bigEndian) {
            return bigEndian?
                  ((short) ((buffer[byteOffset]<<8) |
(buffer[byteOffset+1] & 0xFF))):
                  ((short) ((buffer[byteOffset+1]<<8) |
(buffer[byteOffset] & 0xFF)));
      }

      /**
       * Converts 2 bytes to a signed integer sample with 16bit range.
       * <p> This is a reference function.
       */
      public static int bytesToInt16(byte highByte, byte lowByte) {
            return (highByte<<8) | (lowByte & 0xFF);
      }

      /**
       * Converts 2 successive bytes starting at
<code>byteOffset</code> in
       * <code>buffer</code> to a signed integer sample with 16bit
range.
       * <p>
       * For little endian, buffer[byteOffset] is interpreted as low
byte,
       * whereas it is interpreted as high byte in big endian.
       * <p> This is a reference function.
       */
      public static int bytesToInt16(byte[] buffer, int byteOffset,
boolean bigEndian) {
            return bigEndian?
                  ((buffer[byteOffset]<<8) | (buffer[byteOffset+1] &
0xFF)):
                  ((buffer[byteOffset+1]<<8) | (buffer[byteOffset] &
0xFF));
      }

      /**
```

```
       * Converts 3 successive bytes starting at
<code>byteOffset</code> in
       * <code>buffer</code> to a signed integer sample with 24bit
range.
       * <p>
       * For little endian, buffer[byteOffset] is interpreted as lowest
byte,
       * whereas it is interpreted as highest byte in big endian.
       * <p> This is a reference function.
       */
      public static int bytesToInt24(byte[] buffer, int byteOffset,
boolean bigEndian) {
            return bigEndian?
                  ((buffer[byteOffset]<<16)             // let Java
handle sign-bit
                   | ((buffer[byteOffset+1] & 0xFF)<<8) // inhibit
sign-bit handling
                   | (buffer[byteOffset+2] & 0xFF)):
                  ((buffer[byteOffset+2]<<16)           // let Java
handle sign-bit
                   | ((buffer[byteOffset+1] & 0xFF)<<8) // inhibit
sign-bit handling
                   | (buffer[byteOffset] & 0xFF));
      }

      /**
       * Converts a 4 successive bytes starting at
<code>byteOffset</code> in
       * <code>buffer</code> to a signed 32bit integer sample.
       * <p>
       * For little endian, buffer[byteOffset] is interpreted as lowest
byte,
       * whereas it is interpreted as highest byte in big endian.
       * <p> This is a reference function.
       */
      public static int bytesToInt32(byte[] buffer, int byteOffset,
boolean bigEndian) {
            return bigEndian?
                  ((buffer[byteOffset]<<24)             // let Java
handle sign-bit
                   | ((buffer[byteOffset+1] & 0xFF)<<16) // inhibit
sign-bit handling
                   | ((buffer[byteOffset+2] & 0xFF)<<8)  // inhibit
sign-bit handling
                   | (buffer[byteOffset+3] & 0xFF)):
                  ((buffer[byteOffset+3]<<24)           // let Java
handle sign-bit
                   | ((buffer[byteOffset+2] & 0xFF)<<16) // inhibit
sign-bit handling
                   | ((buffer[byteOffset+1] & 0xFF)<<8)  // inhibit
sign-bit handling
                   | (buffer[byteOffset] & 0xFF));
      }


      /**
```

```
       * Converts a sample of type <code>short</code> to 2 bytes in an
array.
       * <code>sample</code> is interpreted as signed (as Java does).
       * <p>
       * For little endian, buffer[byteOffset] is filled with low byte
of sample,
       * and buffer[byteOffset+1] is filled with high byte of sample.
       * <p> For big endian, this is reversed.
       * <p> This is a reference function.
       */
      public static void shortToBytes16(short sample, byte[] buffer,
int byteOffset, boolean bigEndian) {
            intToBytes16(sample, buffer, byteOffset, bigEndian);
      }

      /**
       * Converts a 16 bit sample of type <code>int</code> to 2 bytes
in an array.
       * <code>sample</code> is interpreted as signed (as Java does).
       * <p>
       * For little endian, buffer[byteOffset] is filled with low byte
of sample,
       * and buffer[byteOffset+1] is filled with high byte of sample +
sign bit.
       * <p> For big endian, this is reversed.
       * <p> Before calling this function, it should be assured that
<code>sample</code>
       * is in the 16bit range - it will not be clipped.
       * <p> This is a reference function.
       */
      public static void intToBytes16(int sample, byte[] buffer, int
byteOffset, boolean bigEndian) {
            if (bigEndian) {
                  buffer[byteOffset++]=(byte) (sample >> 8);
                  buffer[byteOffset]=(byte) (sample & 0xFF);
            } else {
                  buffer[byteOffset++]=(byte) (sample & 0xFF);
                  buffer[byteOffset]=(byte) (sample >> 8);
            }
      }

      /**
       * Converts a 24 bit sample of type <code>int</code> to 3 bytes
in an array.
       * <code>sample</code> is interpreted as signed (as Java does).
       * <p>
       * For little endian, buffer[byteOffset] is filled with low byte
of sample,
       * and buffer[byteOffset+2] is filled with the high byte of
sample + sign bit.
       * <p> For big endian, this is reversed.
       * <p> Before calling this function, it should be assured that
<code>sample</code>
       * is in the 24bit range - it will not be clipped.
       * <p> This is a reference function.
       */
```

```
      public static void intToBytes24(int sample, byte[] buffer, int
byteOffset, boolean bigEndian) {
            if (bigEndian) {
                  buffer[byteOffset++]=(byte) (sample >> 16);
                  buffer[byteOffset++]=(byte) ((sample >>> 8) & 0xFF);
                  buffer[byteOffset]=(byte) (sample & 0xFF);
            } else {
                  buffer[byteOffset++]=(byte) (sample & 0xFF);
                  buffer[byteOffset++]=(byte) ((sample >>> 8) & 0xFF);
                  buffer[byteOffset]=(byte) (sample >> 16);
            }
      }


      /**
       * Converts a 32 bit sample of type <code>int</code> to 4 bytes
in an array.
       * <code>sample</code> is interpreted as signed (as Java does).
       * <p>
       * For little endian, buffer[byteOffset] is filled with lowest
byte of sample,
       * and buffer[byteOffset+3] is filled with the high byte of
sample + sign bit.
       * <p> For big endian, this is reversed.
       * <p> This is a reference function.
       */
      public static void intToBytes32(int sample, byte[] buffer, int
byteOffset, boolean bigEndian) {
            if (bigEndian) {
                  buffer[byteOffset++]=(byte) (sample >> 24);
                  buffer[byteOffset++]=(byte) ((sample >>> 16) & 0xFF);
                  buffer[byteOffset++]=(byte) ((sample >>> 8) & 0xFF);
                  buffer[byteOffset]=(byte) (sample & 0xFF);
            } else {
                  buffer[byteOffset++]=(byte) (sample & 0xFF);
                  buffer[byteOffset++]=(byte) ((sample >>> 8) & 0xFF);
                  buffer[byteOffset++]=(byte) ((sample >>> 16) & 0xFF);
                  buffer[byteOffset]=(byte) (sample >> 24);
            }
      }


      ////////////////////// ULAW
//////////////////////////////////////////

      private static final boolean ZEROTRAP=true;
      private static final short BIAS=0x84;
      private static final int CLIP=32635;
      private static final int exp_lut1[] ={
          0,0,1,1,2,2,2,2,3,3,3,3,3,3,3,3,
          4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
          5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
          5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
          6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
          6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
          6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
          6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
```

```
            7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
            7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
            7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
            7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
            7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
            7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
            7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
            7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7
        };


        /**
         * Converts a linear signed 16bit sample to a uLaw byte.
         * Ported to Java by fb.
         * <BR>Originally by:<BR>
         * Craig Reese: IDA/Supercomputing Research Center <BR>
         * Joe Campbell: Department of Defense <BR>
         * 29 September 1989 <BR>
         */
        public static byte linear2ulaw(int sample) {
              int sign, exponent, mantissa, ulawbyte;

              if (sample>32767) sample=32767;
        else if (sample<-32768) sample=-32768;
              /* Get the sample into sign-magnitude. */
              sign = (sample >> 8) & 0x80;    /* set aside the sign */
              if (sign != 0) sample = -sample;    /* get magnitude */
              if (sample > CLIP) sample = CLIP;    /* clip the magnitude
*/

              /* Convert from 16 bit linear to ulaw. */
              sample = sample + BIAS;
              exponent = exp_lut1[(sample >> 7) & 0xFF];
              mantissa = (sample >> (exponent + 3)) & 0x0F;
              ulawbyte = ~(sign | (exponent << 4) | mantissa);
              if (ZEROTRAP)
                    if (ulawbyte == 0) ulawbyte = 0x02;  /* optional
CCITT trap */
              return((byte) ulawbyte);
        }

        /* u-law to linear conversion table */
        private static short[] u2l = {
            -32124, -31100, -30076, -29052, -28028, -27004, -25980, -
24956,
            -23932, -22908, -21884, -20860, -19836, -18812, -17788, -
16764,
            -15996, -15484, -14972, -14460, -13948, -13436, -12924, -
12412,
            -11900, -11388, -10876, -10364, -9852, -9340, -8828, -8316,
            -7932, -7676, -7420, -7164, -6908, -6652, -6396, -6140,
            -5884, -5628, -5372, -5116, -4860, -4604, -4348, -4092,
            -3900, -3772, -3644, -3516, -3388, -3260, -3132, -3004,
            -2876, -2748, -2620, -2492, -2364, -2236, -2108, -1980,
            -1884, -1820, -1756, -1692, -1628, -1564, -1500, -1436,
            -1372, -1308, -1244, -1180, -1116, -1052, -988, -924,
            -876, -844, -812, -780, -748, -716, -684, -652,
```

```
            -620, -588, -556, -524, -492, -460, -428, -396,
            -372, -356, -340, -324, -308, -292, -276, -260,
            -244, -228, -212, -196, -180, -164, -148, -132,
            -120, -112, -104, -96, -88, -80, -72, -64,
            -56, -48, -40, -32, -24, -16, -8, 0,
            32124, 31100, 30076, 29052, 28028, 27004, 25980, 24956,
            23932, 22908, 21884, 20860, 19836, 18812, 17788, 16764,
            15996, 15484, 14972, 14460, 13948, 13436, 12924, 12412,
            11900, 11388, 10876, 10364, 9852, 9340, 8828, 8316,
            7932, 7676, 7420, 7164, 6908, 6652, 6396, 6140,
            5884, 5628, 5372, 5116, 4860, 4604, 4348, 4092,
            3900, 3772, 3644, 3516, 3388, 3260, 3132, 3004,
            2876, 2748, 2620, 2492, 2364, 2236, 2108, 1980,
            1884, 1820, 1756, 1692, 1628, 1564, 1500, 1436,
            1372, 1308, 1244, 1180, 1116, 1052, 988, 924,
            876, 844, 812, 780, 748, 716, 684, 652,
            620, 588, 556, 524, 492, 460, 428, 396,
            372, 356, 340, 324, 308, 292, 276, 260,
            244, 228, 212, 196, 180, 164, 148, 132,
            120, 112, 104, 96, 88, 80, 72, 64,
            56, 48, 40, 32, 24, 16, 8, 0
      };
      public static short ulaw2linear(byte ulawbyte) {
            return u2l[ulawbyte & 0xFF];
      }



      /**
       * Converts a buffer of signed 16bit big endian samples to uLaw.
       * The uLaw bytes overwrite the original 16 bit values.
       * The first byte-offset of the uLaw bytes is byteOffset.
       * It will be written sampleCount/2 bytes.
       */
      public static void pcm162ulaw(byte[] buffer, int byteOffset, int
sampleCount, boolean bigEndian) {
            int shortIndex=byteOffset;
            int ulawIndex=shortIndex;
            if (bigEndian) {
                  while (sampleCount>0) {
                        buffer[ulawIndex++]=linear2ulaw

(bytesToInt16(buffer[shortIndex], buffer[shortIndex+1]));
                        shortIndex++;
                        shortIndex++;
                        sampleCount--;
                  }
            } else {
                  while (sampleCount>0) {
                        buffer[ulawIndex++]=linear2ulaw

(bytesToInt16(buffer[shortIndex+1], buffer[shortIndex]));
                        shortIndex++;
                        shortIndex++;
                        sampleCount--;
                  }
            }
```

```
	}

	/**
	 * Fills outBuffer with ulaw samples.
	 * reading starts from inBuffer[inByteOffset].
	 * writing starts at outBuffer[outByteOffset].
	 * There will be sampleCount*2 bytes read from inBuffer;
	 * There will be sampleCount <B>bytes</B> written to outBuffer.
	 */
	public static void pcm162ulaw(byte[] inBuffer, int inByteOffset,
	                              byte[] outBuffer, int
outByteOffset,
	                              int sampleCount, boolean bigEndian)
{
		int shortIndex=inByteOffset;
		int ulawIndex=outByteOffset;
		if (bigEndian) {
			while (sampleCount>0) {
				outBuffer[ulawIndex++]=linear2ulaw

(bytesToInt16(inBuffer[shortIndex], inBuffer[shortIndex+1]));
				shortIndex++;
				shortIndex++;
				sampleCount--;
			}
		} else {
			while (sampleCount>0) {
				outBuffer[ulawIndex++]=linear2ulaw

(bytesToInt16(inBuffer[shortIndex+1], inBuffer[shortIndex]));
				shortIndex++;
				shortIndex++;
				sampleCount--;
			}
		}
	}

	// TODO: either direct 8bit pcm to ulaw, or better conversion
from 8bit to 16bit
	/**
	 * Converts a buffer of 8bit samples to uLaw.
	 * The uLaw bytes overwrite the original 8 bit values.
	 * The first byte-offset of the uLaw bytes is byteOffset.
	 * It will be written sampleCount bytes.
	 */
	public static void pcm82ulaw(byte[] buffer, int byteOffset, int
sampleCount, boolean signed) {
		sampleCount+=byteOffset;
		if (signed) {
			for (int i=byteOffset; i<sampleCount; i++) {
				buffer[i]=linear2ulaw(buffer[i] << 8);
			}
		} else {
			for (int i=byteOffset; i<sampleCount; i++) {
				buffer[i]=linear2ulaw(((byte) (buffer[i]+128))
<< 8);
			}
```

```
            }
      }

      /**
       * Fills outBuffer with ulaw samples.
       * reading starts from inBuffer[inByteOffset].
       * writing starts at outBuffer[outByteOffset].
       * There will be sampleCount <B>bytes</B> written to outBuffer.
       */
      public static void pcm82ulaw(byte[] inBuffer, int inByteOffset,
                                   byte[] outBuffer, int outByteOffset,
int sampleCount, boolean signed) {
            int ulawIndex=outByteOffset;
            int pcmIndex=inByteOffset;
            if (signed) {
                  while (sampleCount>0) {

      outBuffer[ulawIndex++]=linear2ulaw(inBuffer[pcmIndex++] << 8);
                        sampleCount--;
                  }
            } else {
                  while (sampleCount>0) {
                        outBuffer[ulawIndex++]=linear2ulaw(((byte)
(inBuffer[pcmIndex++]+128)) << 8);
                        sampleCount--;
                  }
            }
      }

      /**
      * Fills outBuffer with pcm signed 16 bit samples.
      * reading starts from inBuffer[inByteOffset].
      * writing starts at outBuffer[outByteOffset].
      * There will be sampleCount bytes read from inBuffer;
      * There will be sampleCount*2 bytes written to outBuffer.
      */
      public static void ulaw2pcm16(byte[] inBuffer, int inByteOffset,
                                    byte[] outBuffer, int
outByteOffset,
                                    int sampleCount, boolean bigEndian)
{
            int shortIndex=outByteOffset;
            int ulawIndex=inByteOffset;
            while (sampleCount>0) {
                  intToBytes16
                  (u2l[inBuffer[ulawIndex++] & 0xFF], outBuffer,
shortIndex++, bigEndian);
                  shortIndex++;
                  sampleCount--;
            }
      }


      // TODO: either direct 8bit pcm to ulaw, or better conversion
from 8bit to 16bit
      /**
       * Inplace-conversion of a ulaw buffer to 8bit samples.
```

```
       * The 8bit bytes overwrite the original ulaw values.
       * The first byte-offset of the uLaw bytes is byteOffset.
       * It will be written sampleCount bytes.
       */
      public static void ulaw2pcm8(byte[] buffer, int byteOffset, int
sampleCount, boolean signed) {
            sampleCount+=byteOffset;
            if (signed) {
                  for (int i=byteOffset; i<sampleCount; i++) {
                        buffer[i]=(byte) ((u2l[buffer[i] & 0xFF] >> 8)
& 0xFF);
                  }
            } else {
                  for (int i=byteOffset; i<sampleCount; i++) {
                        buffer[i]=(byte) ((u2l[buffer[i] &
0xFF]>>8)+128);
                  }
            }
      }

      /**
       * Fills outBuffer with ulaw samples.
       * reading starts from inBuffer[inByteOffset].
       * writing starts at outBuffer[outByteOffset].
       * There will be sampleCount <B>bytes</B> written to outBuffer.
       */
      public static void ulaw2pcm8(byte[] inBuffer, int inByteOffset,
                                   byte[] outBuffer, int outByteOffset,
int sampleCount, boolean signed) {
            int ulawIndex=inByteOffset;
            int pcmIndex=outByteOffset;
            if (signed) {
                  while (sampleCount>0) {
                        outBuffer[pcmIndex++]=
                            (byte) ((u2l[inBuffer[ulawIndex++] & 0xFF]
>> 8) & 0xFF);
                        sampleCount--;
                  }
            } else {
                  while (sampleCount>0) {
                        outBuffer[pcmIndex++]=
                            (byte) ((u2l[inBuffer[ulawIndex++] &
0xFF]>>8)+128);
                        sampleCount--;
                  }
            }
      }


      ////////////////////// ALAW ////////////////////////////


      /*
       * This source code is a product of Sun Microsystems, Inc. and is
provided
       * for unrestricted use.  Users may copy or modify this source
code without
```

```
       * charge.
       *
       * linear2alaw() - Convert a 16-bit linear PCM value to 8-bit A-
law
       *
       * linear2alaw() accepts an 16-bit integer and encodes it as A-
law data.
       *
       *          Linear Input Code Compressed Code
       *    ------------------------      ---------------
       *    0000000wxyza                  000wxyz
       *    0000001wxyza                  001wxyz
       *    000001wxyzab                  010wxyz
       *    00001wxyzabc                  011wxyz
       *    0001wxyzabcd                  100wxyz
       *    001wxyzabcde                  101wxyz
       *    01wxyzabcdef                  110wxyz
       *    1wxyzabcdefg                  111wxyz
       *
       * For further information see John C. Bellamy's Digital
Telephony, 1982,
       * John Wiley & Sons, pps 98-111 and 472-476.
       */
      private static final byte QUANT_MASK = 0xf;           /*
Quantization field mask. */
      private static final byte SEG_SHIFT = 4;        /* Left shift for
segment number. */
      private static final short[] seg_end = {
          0xFF, 0x1FF, 0x3FF, 0x7FF, 0xFFF, 0x1FFF, 0x3FFF, 0x7FFF
      };

      public static byte linear2alaw(short pcm_val)   /* 2's complement
(16-bit range) */
      {
            byte  mask;
            byte  seg=8;
            byte  aval;

            if (pcm_val >= 0) {
                  mask = (byte) 0xD5;           /* sign (7th) bit = 1
*/
            } else {
                  mask = 0x55;            /* sign bit = 0 */
                  pcm_val = (short) (-pcm_val - 8);
            }

            /* Convert the scaled magnitude to segment number. */
            for (int i = 0; i < 8; i++) {
                  if (pcm_val <= seg_end[i]) {
                        seg=(byte) i;
                        break;
                  }
            }

            /* Combine the sign, segment, and quantization bits. */
            if (seg >= 8)           /* out of range, return maximum
value. */
```

```
                return (byte) ((0x7F ^ mask) & 0xFF);
            else {
                aval = (byte) (seg << SEG_SHIFT);
                if (seg < 2)
                    aval |= (pcm_val >> 4) & QUANT_MASK;
                else
                    aval |= (pcm_val >> (seg + 3)) & QUANT_MASK;
                return (byte) ((aval ^ mask) & 0xFF);
            }
        }

        private static short[] a2l = {
            -5504, -5248, -6016, -5760, -4480, -4224, -4992, -4736,
            -7552, -7296, -8064, -7808, -6528, -6272, -7040, -6784,
            -2752, -2624, -3008, -2880, -2240, -2112, -2496, -2368,
            -3776, -3648, -4032, -3904, -3264, -3136, -3520, -3392,
            -22016, -20992, -24064, -23040, -17920, -16896, -19968, -
18944,
            -30208, -29184, -32256, -31232, -26112, -25088, -28160, -
27136,
            -11008, -10496, -12032, -11520, -8960, -8448, -9984, -9472,
            -15104, -14592, -16128, -15616, -13056, -12544, -14080, -
13568,
            -344, -328, -376, -360, -280, -264, -312, -296,
            -472, -456, -504, -488, -408, -392, -440, -424,
            -88, -72, -120, -104, -24, -8, -56, -40,
            -216, -200, -248, -232, -152, -136, -184, -168,
            -1376, -1312, -1504, -1440, -1120, -1056, -1248, -1184,
            -1888, -1824, -2016, -1952, -1632, -1568, -1760, -1696,
            -688, -656, -752, -720, -560, -528, -624, -592,
            -944, -912, -1008, -976, -816, -784, -880, -848,
            5504, 5248, 6016, 5760, 4480, 4224, 4992, 4736,
            7552, 7296, 8064, 7808, 6528, 6272, 7040, 6784,
            2752, 2624, 3008, 2880, 2240, 2112, 2496, 2368,
            3776, 3648, 4032, 3904, 3264, 3136, 3520, 3392,
            22016, 20992, 24064, 23040, 17920, 16896, 19968, 18944,
            30208, 29184, 32256, 31232, 26112, 25088, 28160, 27136,
            11008, 10496, 12032, 11520, 8960, 8448, 9984, 9472,
            15104, 14592, 16128, 15616, 13056, 12544, 14080, 13568,
            344, 328, 376, 360, 280, 264, 312, 296,
            472, 456, 504, 488, 408, 392, 440, 424,
            88, 72, 120, 104, 24, 8, 56, 40,
            216, 200, 248, 232, 152, 136, 184, 168,
            1376, 1312, 1504, 1440, 1120, 1056, 1248, 1184,
            1888, 1824, 2016, 1952, 1632, 1568, 1760, 1696,
            688, 656, 752, 720, 560, 528, 624, 592,
            944, 912, 1008, 976, 816, 784, 880, 848
        };

        public static short alaw2linear(byte ulawbyte) {
            return a2l[ulawbyte & 0xFF];
        }

        /**
         * Converts a buffer of signed 16bit big endian samples to uLaw.
         * The uLaw bytes overwrite the original 16 bit values.
         * The first byte-offset of the uLaw bytes is byteOffset.
```

```
       * It will be written sampleCount/2 bytes.
       */
      public static void pcm162alaw(byte[] buffer, int byteOffset, int
sampleCount, boolean bigEndian) {
            int shortIndex=byteOffset;
            int alawIndex=shortIndex;
            if (bigEndian) {
                  while (sampleCount>0) {
                        buffer[alawIndex++]=
                            linear2alaw(bytesToShort16
                                        (buffer[shortIndex],
buffer[shortIndex+1]));
                        shortIndex++;
                        shortIndex++;
                        sampleCount--;
                  }
            } else {
                  while (sampleCount>0) {
                        buffer[alawIndex++]=
                            linear2alaw(bytesToShort16
                                        (buffer[shortIndex+1],
buffer[shortIndex]));
                        shortIndex++;
                        shortIndex++;
                        sampleCount--;
                  }
            }
      }

      /**
       * Fills outBuffer with alaw samples.
       * reading starts from inBuffer[inByteOffset].
       * writing starts at outBuffer[outByteOffset].
       * There will be sampleCount*2 bytes read from inBuffer;
       * There will be sampleCount <B>bytes</B> written to outBuffer.
       */
      public static void pcm162alaw(byte[] inBuffer, int inByteOffset,
                                    byte[] outBuffer, int
outByteOffset, int sampleCount, boolean bigEndian) {
            int shortIndex=inByteOffset;
            int alawIndex=outByteOffset;
            if (bigEndian) {
                  while (sampleCount>0) {
                        outBuffer[alawIndex++]=linear2alaw

(bytesToShort16(inBuffer[shortIndex], inBuffer[shortIndex+1]));
                        shortIndex++;
                        shortIndex++;
                        sampleCount--;
                  }
            } else {
                  while (sampleCount>0) {
                        outBuffer[alawIndex++]=linear2alaw

(bytesToShort16(inBuffer[shortIndex+1], inBuffer[shortIndex]));
                        shortIndex++;
                        shortIndex++;
```

```
                        sampleCount--;
                  }
            }
      }

      /**
       * Converts a buffer of 8bit samples to alaw.
       * The alaw bytes overwrite the original 8 bit values.
       * The first byte-offset of the aLaw bytes is byteOffset.
       * It will be written sampleCount bytes.
       */
      public static void pcm82alaw(byte[] buffer, int byteOffset, int
sampleCount, boolean signed) {
            sampleCount+=byteOffset;
            if (signed) {
                  for (int i=byteOffset; i<sampleCount; i++) {
                        buffer[i]=linear2alaw((short) (buffer[i] <<
8));
                  }
            } else {
                  for (int i=byteOffset; i<sampleCount; i++) {
                        buffer[i]=linear2alaw((short) (((byte)
(buffer[i]+128)) << 8));
                  }
            }
      }

      /**
       * Fills outBuffer with alaw samples.
       * reading starts from inBuffer[inByteOffset].
       * writing starts at outBuffer[outByteOffset].
       * There will be sampleCount <B>bytes</B> written to outBuffer.
       */
      public static void pcm82alaw(byte[] inBuffer, int inByteOffset,
                                   byte[] outBuffer, int outByteOffset,
int sampleCount, boolean signed) {
            int alawIndex=outByteOffset;
            int pcmIndex=inByteOffset;
            if (signed) {
                  while (sampleCount>0) {
                        outBuffer[alawIndex++]=
                            linear2alaw((short) (inBuffer[pcmIndex++]
<< 8));
                        sampleCount--;
                  }
            } else {
                  while (sampleCount>0) {
                        outBuffer[alawIndex++]=
                            linear2alaw((short) (((byte)
(inBuffer[pcmIndex++]+128)) << 8));
                        sampleCount--;
                  }
            }
      }
```

```
      /**
       * Converts an alaw buffer to 8bit pcm samples
       * The 8bit bytes overwrite the original alaw values.
       * The first byte-offset of the aLaw bytes is byteOffset.
       * It will be written sampleCount bytes.
       */
      public static void alaw2pcm8(byte[] buffer, int byteOffset, int
sampleCount, boolean signed) {
            sampleCount+=byteOffset;
            if (signed) {
                  for (int i=byteOffset; i<sampleCount; i++) {
                        buffer[i]=(byte) ((a2l[buffer[i] & 0xFF] >> 8)
& 0xFF);
                  }
            } else {
                  for (int i=byteOffset; i<sampleCount; i++) {
                        buffer[i]=(byte) ((a2l[buffer[i] &
0xFF]>>8)+128);
                  }
            }
      }

      /**
       * Fills outBuffer with alaw samples.
       * reading starts from inBuffer[inByteOffset].
       * writing starts at outBuffer[outByteOffset].
       * There will be sampleCount <B>bytes</B> written to outBuffer.
       */
      public static void alaw2pcm8(byte[] inBuffer, int inByteOffset,
                                   byte[] outBuffer, int outByteOffset,
int sampleCount, boolean signed) {
            int alawIndex=inByteOffset;
            int pcmIndex=outByteOffset;
            if (signed) {
                  while (sampleCount>0) {
                        outBuffer[pcmIndex++]=
                            (byte) ((a2l[inBuffer[alawIndex++] & 0xFF]
>> 8) & 0xFF);
                        sampleCount--;
                  }
            } else {
                  while (sampleCount>0) {
                        outBuffer[pcmIndex++]=
                            (byte) ((a2l[inBuffer[alawIndex++] &
0xFF]>>8)+128);
                        sampleCount--;
                  }
            }
      }

      /**
       * Fills outBuffer with pcm signed 16 bit samples.
       * reading starts from inBuffer[inByteOffset].
       * writing starts at outBuffer[outByteOffset].
       * There will be sampleCount bytes read from inBuffer;
       * There will be sampleCount*2 bytes written to outBuffer.
       */
```

```
      public static void alaw2pcm16(byte[] inBuffer, int inByteOffset,
                                    byte[] outBuffer, int
outByteOffset,
                                    int sampleCount, boolean bigEndian)
{
            int shortIndex=outByteOffset;
            int alawIndex=inByteOffset;
            while (sampleCount>0) {
                  intToBytes16
                  (a2l[inBuffer[alawIndex++] & 0xFF], outBuffer,
shortIndex++, bigEndian);
                  shortIndex++;
                  sampleCount--;
            }
      }

      ////////////////////// cross conversion alaw <-> ulaw
//////////////////////////////////////

      private static byte[] u2a = {
          -86, -85, -88, -87, -82, -81, -84, -83, -94, -93, -96, -95, -
90, -89, -92, -91,
          -70, -69, -72, -71, -66, -65, -68, -67, -78, -77, -80, -79, -
74, -73, -76, -75,
          -118, -117, -120, -119, -114, -113, -116, -115, -126, -125, -
128, -127, -122, -121, -124, -123,
          -101, -104, -103, -98, -97, -100, -99, -110, -109, -112, -
111, -106, -105, -108, -107, -22,
          -24, -23, -18, -17, -20, -19, -30, -29, -32, -31, -26, -25, -
28, -27, -6, -8,
          -2, -1, -4, -3, -14, -13, -16, -15, -10, -9, -12, -11, -53, -
55, -49, -51,
          -62, -61, -64, -63, -58, -57, -60, -59, -38, -37, -40, -39, -
34, -33, -36, -35,
          -46, -46, -45, -45, -48, -48, -47, -47, -42, -42, -41, -41, -
44, -44, -43, -43,
          42, 43, 40, 41, 46, 47, 44, 45, 34, 35, 32, 33, 38, 39, 36,
37,
          58, 59, 56, 57, 62, 63, 60, 61, 50, 51, 48, 49, 54, 55, 52,
53,
          10, 11, 8, 9, 14, 15, 12, 13, 2, 3, 0, 1, 6, 7, 4, 5,
          27, 24, 25, 30, 31, 28, 29, 18, 19, 16, 17, 22, 23, 20, 21,
106,
          104, 105, 110, 111, 108, 109, 98, 99, 96, 97, 102, 103, 100,
101, 122, 120,
          126, 127, 124, 125, 114, 115, 112, 113, 118, 119, 116, 117,
75, 73, 79, 77,
          66, 67, 64, 65, 70, 71, 68, 69, 90, 91, 88, 89, 94, 95, 92,
93,
          82, 82, 83, 83, 80, 80, 81, 81, 86, 86, 87, 87, 84, 84, 85,
85,
      };

      public static byte ulaw2alaw(byte sample) {
            return u2a[sample & 0xFF];
      }
```

```
      /**
       * Converts a buffer of uLaw samples to aLaw.
       */
      public static void ulaw2alaw(byte[] buffer, int byteOffset, int
sampleCount) {
            sampleCount+=byteOffset;
            for (int i=byteOffset; i<sampleCount; i++) {
                  buffer[i]=u2a[buffer[i] & 0xFF];
            }
      }

      /**
       * Fills outBuffer with alaw samples.
       */
      public static void ulaw2alaw(byte[] inBuffer, int inByteOffset,
                                   byte[] outBuffer, int outByteOffset,
int sampleCount) {
            int ulawIndex=outByteOffset;
            int alawIndex=inByteOffset;
            while (sampleCount>0) {
                  outBuffer[alawIndex++]=u2a[inBuffer[ulawIndex++] &
0xFF];
                  sampleCount--;
            }
      }

      private static byte[] a2u = {
          -86, -85, -88, -87, -82, -81, -84, -83, -94, -93, -96, -95, -
90, -89, -92, -91,
          -71, -70, -73, -72, -67, -66, -69, -68, -79, -78, -80, -80, -
75, -74, -77, -76,
          -118, -117, -120, -119, -114, -113, -116, -115, -126, -125, -
128, -127, -122, -121, -124, -123,
          -102, -101, -104, -103, -98, -97, -100, -99, -110, -109, -
112, -111, -106, -105, -108, -107,
          -30, -29, -32, -31, -26, -25, -28, -27, -35, -35, -36, -36, -
33, -33, -34, -34,
          -12, -10, -16, -14, -4, -2, -8, -6, -22, -21, -24, -23, -18,
-17, -20, -19,
          -56, -55, -58, -57, -52, -51, -54, -53, -64, -63, -65, -65, -
60, -59, -62, -61,
          -42, -41, -44, -43, -38, -37, -40, -39, -49, -49, -50, -50, -
46, -45, -48, -47,
          42, 43, 40, 41, 46, 47, 44, 45, 34, 35, 32, 33, 38, 39, 36,
37,
          57, 58, 55, 56, 61, 62, 59, 60, 49, 50, 48, 48, 53, 54, 51,
52,
          10, 11, 8, 9, 14, 15, 12, 13, 2, 3, 0, 1, 6, 7, 4, 5,
          26, 27, 24, 25, 30, 31, 28, 29, 18, 19, 16, 17, 22, 23, 20,
21,
          98, 99, 96, 97, 102, 103, 100, 101, 93, 93, 92, 92, 95, 95,
94, 94,
          116, 118, 112, 114, 124, 126, 120, 122, 106, 107, 104, 105,
110, 111, 108, 109,
          72, 73, 70, 71, 76, 77, 74, 75, 64, 65, 63, 63, 68, 69, 66,
67,
```

```
          86, 87, 84, 85, 90, 91, 88, 89, 79, 79, 78, 78, 82, 83, 80,
81,
      };

      public static byte alaw2ulaw(byte sample) {
            return a2u[sample & 0xFF];
      }

      /**
       * Converts a buffer of aLaw samples to uLaw.
       * The uLaw bytes overwrite the original aLaw values.
       * The first byte-offset of the uLaw bytes is byteOffset.
       * It will be written sampleCount bytes.
       */
      public static void alaw2ulaw(byte[] buffer, int byteOffset, int
sampleCount) {
            sampleCount+=byteOffset;
            for (int i=byteOffset; i<sampleCount; i++) {
                  buffer[i]=a2u[buffer[i] & 0xFF];
            }
      }

      /**
       * Fills outBuffer with ulaw samples.
       * reading starts from inBuffer[inByteOffset].
       * writing starts at outBuffer[outByteOffset].
       * There will be sampleCount <B>bytes</B> written to outBuffer.
       */
      public static void alaw2ulaw(byte[] inBuffer, int inByteOffset,
                                   byte[] outBuffer, int outByteOffset,
int sampleCount) {
            int ulawIndex=outByteOffset;
            int alawIndex=inByteOffset;
            while (sampleCount>0) {
                  outBuffer[ulawIndex++]=a2u[inBuffer[alawIndex++] &
0xFF];
                  sampleCount--;
            }
      }


      ////////////////////// high level methods
//////////////////////////////////////////////

      /*
       *    !! Here, unlike other functions in this class, the length
is
       *    in bytes rather than samples !!
       */
      public static void changeOrderOrSign(byte[] buffer, int nOffset,
                                           int nByteLength, int
nBytesPerSample) {
            switch (nBytesPerSample) {
            case 1:
                  convertSign8(buffer, nOffset, nByteLength);
                  break;
```

```
            case 2:
                  swapOrder16(buffer, nOffset, nByteLength / 2);
                  break;

            case 3:
                  swapOrder24(buffer, nOffset, nByteLength / 3);
                  break;

            case 4:
                  swapOrder32(buffer, nOffset, nByteLength / 4);
                  break;
            }
      }



      /*
       *    !! Here, unlike other functions in this class, the length
is
       *    in bytes rather than samples !!
       */
      public static void changeOrderOrSign(
          byte[] inBuffer, int nInOffset,
          byte[] outBuffer, int nOutOffset,
          int nByteLength, int nBytesPerSample) {
            switch (nBytesPerSample) {
            case 1:
                  convertSign8(
                      inBuffer, nInOffset,
                      outBuffer, nOutOffset,
                      nByteLength);
                  break;

            case 2:
                  swapOrder16(
                      inBuffer, nInOffset,
                      outBuffer, nOutOffset,
                      nByteLength / 2);
                  break;

            case 3:
                  swapOrder24(
                      inBuffer, nInOffset,
                      outBuffer, nOutOffset,
                      nByteLength / 3);
                  break;

            case 4:
                  swapOrder32(
                      inBuffer, nInOffset,
                      outBuffer, nOutOffset,
                      nByteLength / 4);
                  break;
            }
      }
```

```
      ///////////////// Annexe: how the arrays were created.
////////////////////////////////

      /*
       * Converts a uLaw byte to a linear signed 16bit sample.
       * Ported to Java by fb.
       * <BR>Originally by:<BR>
       *
       * Craig Reese: IDA/Supercomputing Research Center <BR>
       * 29 September 1989 <BR>
       *
       * References: <BR>
       * <OL>
       * <LI>CCITT Recommendation G.711  (very difficult to
follow)</LI>
       * <LI>MIL-STD-188-113,"Interoperability and Performance
Standards
       *     for Analog-to_Digital Conversion Techniques,"
       *     17 February 1987</LI>
       * </OL>
       */
      /*
      private static final int exp_lut2[] = {
      0,132,396,924,1980,4092,8316,16764
};

      public static short _ulaw2linear(int ulawbyte) {
      int sign, exponent, mantissa, sample;

      ulawbyte = ~ulawbyte;
      sign = (ulawbyte & 0x80);
      exponent = (ulawbyte >> 4) & 0x07;
      mantissa = ulawbyte & 0x0F;
      sample = exp_lut2[exponent] + (mantissa << (exponent + 3));
      if (sign != 0) sample = -sample;
      return((short) sample);
}*/


      /* u- to A-law conversions: copied from CCITT G.711
specifications */
      /*
      private static byte[] _u2a = {
      1,    1,    2,    2,    3,    3,    4,    4,
      5,    5,    6,    6,    7,    7,    8,    8,
      9,    10,   11,   12,   13,   14,   15,   16,
      17,   18,   19,   20,   21,   22,   23,   24,
      25,   27,   29,   31,   33,   34,   35,   36,
      37,   38,   39,   40,   41,   42,   43,   44,
      46,   48,   49,   50,   51,   52,   53,   54,
      55,   56,   57,   58,   59,   60,   61,   62,
      64,   65,   66,   67,   68,   69,   70,   71,
      72,   73,   74,   75,   76,   77,   78,   79,
      81,   82,   83,   84,   85,   86,   87,   88,
      89,   90,   91,   92,   93,   94,   95,   96,
      97,   98,   99,   100,  101,  102,  103,  104,
      105,  106,  107,  108,  109,  110,  111,  112,
```

```
      113,  114,  115,  116,  117,  118,  119,  120,
      121,  122,  123,  124,  125,  126,  127,  (byte) 128};
      */

      /* u-law to A-law conversion */
      /*
       * This source code is a product of Sun Microsystems, Inc. and is
provided
       * for unrestricted use.  Users may copy or modify this source
code without
       * charge.
       */
      /*
      public static byte _ulaw2alaw(byte sample) {
      sample &= 0xff;
      return (byte) (((sample & 0x80)!=0) ? (0xD5 ^ (_u2a[(0x7F ^
sample) & 0x7F] - 1)) :
           (0x55 ^ (_u2a[(0x7F ^ sample) & 0x7F] - 1)));
}*/

      /* A- to u-law conversions */
      /*
      private static byte[] _a2u = {
      1,    3,    5,    7,    9,    11,   13,   15,
      16,   17,   18,   19,   20,   21,   22,   23,
      24,   25,   26,   27,   28,   29,   30,   31,
      32,   32,   33,   33,   34,   34,   35,   35,
      36,   37,   38,   39,   40,   41,   42,   43,
      44,   45,   46,   47,   48,   48,   49,   49,
      50,   51,   52,   53,   54,   55,   56,   57,
      58,   59,   60,   61,   62,   63,   64,   64,
      65,   66,   67,   68,   69,   70,   71,   72,
      73,   74,   75,   76,   77,   78,   79,   79,
      80,   81,   82,   83,   84,   85,   86,   87,
      88,   89,   90,   91,   92,   93,   94,   95,
      96,   97,   98,   99,   100,  101,  102,  103,
      104,  105,  106,  107,  108,  109,  110,  111,
      112,  113,  114,  115,  116,  117,  118,  119,
      120,  121,  122,  123,  124,  125,  126,  127};
      */

      /*
       * This source code is a product of Sun Microsystems, Inc. and is
provided
       * for unrestricted use.  Users may copy or modify this source
code without
       * charge.
       */
      /*
      public static byte _alaw2ulaw(byte sample) {
      sample &= 0xff;
      return (byte) (((sample & 0x80)!=0) ? (0xFF ^ _a2u[(sample ^
0xD5) & 0x7F]) :
           (0x7F ^ _a2u[(sample ^ 0x55) & 0x7F]));
}

      public static void print_a2u() {
```

```
        System.out.println("\tprivate static byte[] a2u = {");
        for (int i=-128; i<128; i++) {
         if (((i+128) % 16)==0) {
        System.out.print("\t\t");
         }
         byte b=(byte) i;
         System.out.print(_alaw2ulaw(b)+", ");
         if (((i+128) % 16)==15) {
        System.out.println("");
         }
}
        System.out.println("\t};");
}

        public static void print_u2a() {
        System.out.println("\tprivate static byte[] u2a = {");
        for (int i=-128; i<128; i++) {
         if (((i+128) % 16)==0) {
        System.out.print("\t\t");
         }
         byte b=(byte) i;
         System.out.print(_ulaw2alaw(b)+", ");
         if (((i+128) % 16)==15) {
        System.out.println("");
         }
}
        System.out.println("\t};");
}
        */

}


/*** TConversionTool.java ***/
```