```
/*
Live Sound Utility Version 1.0

------------------------------

Bruce Mechtely & Joshua Wurtz

---------------

GUI                 lines: 75-226
Listeners           lines: 227 - 296
Process from File   lines: 323 - 387
Process Realtime    lines: 370 - 445
Capture File        lines: 446 - 484
Processing Methods  lines: 485 - 589
Variable preparation  lines: 590 - 674
*/
import java.io.*;
import javax.sound.sampled.*;
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
import javax.swing.filechooser.*;

public class LiveSoundUtility extends JFrame
```

```
{
  private JTabbedPane tabbedPane;
  private JSpinner echoDelaySpin, echoDepthSpin, echoRepeatSpin, betaPitchSpin, betaRobotSpin;
  private JSpinner flangeMinDelaySpin, flangeMaxDelaySpin, flangeDepthSpin, flangePeriodSpin,
timeSpin;

  private String[] hzArray = {"44100","22050","11025","8000"};
  private String[] windowSizeArray = {"32","64","128","256","512","1024","2048","4096"};
  private JComboBox hzCombo,windowSizeCombo;

  private JButton exit, play;
  private MenuListener menuLstn;
  private ButtonListener buttonLstn;

  private JCheckBox harmonizer;
  private JMenuBar jmb;
  private JMenu menu;
  private JMenuItem menuItemCapToFile, menuItemProcessRealTime, menuItemProcessFromFile;

  private int srcSelected=1, index, windowSize=1024;

  private AudioInputStream aisFromFile;
  private AudioFormat formatFromFile;
  private long frameLengthFromFile;

  private File outFile=null, inFile=null;
```

```
private float fps=0;


private float echoDepth,echoDelay;
private int echoRepeat;


private float flangeMinDelay, flangeMaxDelay, flangeDepth, flangePeriod;
private int flangeMinFrames, flangeMaxFrames, flangeDelay, frameCount=0, frameSkip;
private boolean up = true;


private float pi=(float)Math.PI, beta, scaleA, scaleS;;
private int overlap = 4, hopA, hopS;
private float[] hannWindowA, hannWindowS, prevPhiA, prevPhiS;
private float real,imag,mag,phiA,wBin,deltaPhi,deltaPhiWrap,wTrue,phiS,maxFFT,freq;


private int numWindows=0, writeDelay=0, bufferSize=0;
private float[] inBuffer, outBuffer;
private int readOffset=0, writeOffset=0, chunkSize=0;
private byte[] chunk;
private SourceDataLine sourceLine;
private TargetDataLine targetLine;


public LiveSoundUtility()
{
  JFrame frame = new JFrame("Sound Processing");
  menuLstn = new MenuListener();
```

```
buttonLstn = new ButtonListener();


JPanel hzAndWindowPanel = new JPanel();
hzCombo = new JComboBox(hzArray);
windowSizeCombo = new JComboBox(windowSizeArray);
JLabel hzLabel = new JLabel("HZ:");
JLabel windowSizeLabel = new JLabel("Window Size:");
timeSpin = new JSpinner(new SpinnerNumberModel(2f, 1f, 1800f, 1f));
JLabel timeLabel = new JLabel("Time (sec): ");
hzAndWindowPanel.add(timeLabel);
hzAndWindowPanel.add(timeSpin);
hzAndWindowPanel.add(hzLabel);
hzAndWindowPanel.add(hzCombo);
hzAndWindowPanel.add(windowSizeLabel);
hzAndWindowPanel.add(windowSizeCombo);


tabbedPane = new JTabbedPane();
JComponent echoPanel = createEchoPanel();
tabbedPane.addTab("Echo", null, echoPanel,"Echo Processing");
tabbedPane.setMnemonicAt(0, KeyEvent.VK_1);
JComponent flangePanel = createFlangePanel();
tabbedPane.addTab("Flange", null, flangePanel,"Flange Processing");
tabbedPane.setMnemonicAt(1, KeyEvent.VK_2);
JComponent pitchShiftPanel = createPitchPanel();
tabbedPane.addTab("Pitch Shift", null, pitchShiftPanel,"Pitch Shift Processing");
```

```
  tabbedPane.setMnemonicAt(2, KeyEvent.VK_3);
  JComponent noProcessPanel = createNoProcessPanel();
  tabbedPane.addTab("No Processing", null, noProcessPanel,"No Processing");
  tabbedPane.setMnemonicAt(3, KeyEvent.VK_3);
  JComponent robotVoicePanel = createRobotVoicePanel();
  tabbedPane.addTab("Robot Voice", null, robotVoicePanel,"Robot Voice");
  tabbedPane.setMnemonicAt(4, KeyEvent.VK_4);
  tabbedPane.setPreferredSize(new Dimension(400,125));

  JPanel centerPanel = new JPanel();
  centerPanel.setLayout(new BoxLayout(centerPanel, BoxLayout.Y_AXIS));
  centerPanel.add(hzAndWindowPanel);
  centerPanel.add(tabbedPane);

  frame.add(createMenu(), BorderLayout.PAGE_START);
  frame.add(centerPanel, BorderLayout.CENTER);
  frame.add(createButtons(), BorderLayout.PAGE_END);
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  frame.setLocation(300,150);
  frame.pack();
  frame.setResizable(true);
  frame.setVisible(true);
}
protected JComponent createMenu()
{
```

```
  jmb = new JMenuBar();
  menu = new JMenu("Source");
  menu.setMnemonic(KeyEvent.VK_E);
  menuItemCapToFile = new JMenuItem("Capture to File");
  menuItemCapToFile.setMnemonic(KeyEvent.VK_C);
  menuItemCapToFile.addActionListener(menuLstn);
  menuItemProcessRealTime = new JMenuItem("Process Real Time");
  menuItemProcessRealTime.setMnemonic(KeyEvent.VK_A);
  menuItemProcessRealTime.addActionListener(menuLstn);
  menuItemProcessFromFile = new JMenuItem("Process From File");
  menuItemProcessFromFile.setMnemonic(KeyEvent.VK_P);
  menuItemProcessFromFile.addActionListener(menuLstn);
  menu.add(menuItemCapToFile);
  menu.add(menuItemProcessRealTime);
  menu.add(menuItemProcessFromFile);
  jmb.add(menu);
  return jmb;
}
protected JComponent createEchoPanel()
{
  JPanel panel= new JPanel();
  JPanel echoSelections = new JPanel();
  echoSelections.setLayout(new GridLayout(3,2));
  echoDelaySpin = new JSpinner(new SpinnerNumberModel(500, 0, 2000, 10));
  echoDepthSpin = new JSpinner(new SpinnerNumberModel(0.8f, 0.1f, .99f, 0.01f));
```

```
  echoRepeatSpin = new JSpinner(new SpinnerNumberModel(1, 1, 10, 1));
  JLabel echoDelayLabel = new JLabel("Echo Delay(ms): ");
  JLabel echoDepthLabel = new JLabel("Echo Depth: ");
  JLabel echoRepeatLabel = new JLabel("Number of Echos: ");
  echoSelections.add(echoDelayLabel);
  echoSelections.add(echoDelaySpin);
  echoSelections.add(echoDepthLabel);
  echoSelections.add(echoDepthSpin);
  echoSelections.add(echoRepeatLabel);
  echoSelections.add(echoRepeatSpin);
  panel.add(echoSelections, BorderLayout.CENTER);
  return panel;
}
protected JComponent createFlangePanel()
{
  JPanel panel = new JPanel();
  JPanel flangeSelections = new JPanel();
  flangeSelections.setLayout(new GridLayout(4,2));
  flangeMinDelaySpin = new JSpinner(new SpinnerNumberModel(0, 0, 20, 1));
  flangeMaxDelaySpin = new JSpinner(new SpinnerNumberModel(20, 0, 50, 1));
  flangeDepthSpin = new JSpinner(new SpinnerNumberModel(.8f, 0, 1f, .01f));
  flangePeriodSpin = new JSpinner(new SpinnerNumberModel(2000, 0, 8000, 100));
  JLabel flangeMinDelayLabel = new JLabel("Minimum Delay (ms): ");
  JLabel flangeMaxDelayLabel = new JLabel("Maximum Delay (ms): ");
  JLabel flangeDepthLabel = new JLabel("Flange Depth: ");
```

```
  JLabel flangePeriodLabel = new JLabel("Flange Period: ");
  flangeSelections.add(flangeMinDelayLabel);
  flangeSelections.add(flangeMinDelaySpin);
  flangeSelections.add(flangeMaxDelayLabel);
  flangeSelections.add(flangeMaxDelaySpin);
  flangeSelections.add(flangeDepthLabel);
  flangeSelections.add(flangeDepthSpin);
  flangeSelections.add(flangePeriodLabel);
  flangeSelections.add(flangePeriodSpin);
  panel.add(flangeSelections, BorderLayout.CENTER);
  return panel;
}
protected JComponent createPitchPanel()
{
  JPanel panel = new JPanel();
  JPanel pitchSelections = new JPanel();
  betaPitchSpin = new JSpinner(new SpinnerNumberModel(0.8f, 0.01f, 3.0f, 0.01f));
  harmonizer = new JCheckBox("Harmonizer");
  JLabel pitchBetaLabel = new JLabel("Beta: ");
  pitchSelections.add(harmonizer);
  pitchSelections.add(pitchBetaLabel);
  pitchSelections.add(betaPitchSpin);
  panel.add(pitchSelections, BorderLayout.CENTER);
  return panel;
}
```

```
protected JComponent createNoProcessPanel()
{
  JPanel panel= new JPanel();
  return panel;
}
protected JComponent createRobotVoicePanel()
{
  JPanel panel = new JPanel();
  JPanel robotVoice = new JPanel();
  betaRobotSpin = new JSpinner(new SpinnerNumberModel(0.8f, 0.01f, 3.0f, 0.01f));
  JLabel robotBetaLabel = new JLabel("Beta: ");
  robotVoice.add(robotBetaLabel);
  robotVoice.add(betaRobotSpin);
  panel.add(robotVoice, BorderLayout.CENTER);
  return panel;
}
protected JComponent createButtons()
{
  JPanel panel = new JPanel();
  play = new JButton("Process");
  play.addActionListener(buttonLstn);
  exit = new JButton("Exit");
  exit.addActionListener(buttonLstn);
  play.setEnabled(false);
  exit.setEnabled(true);
```

```
  panel.add(play);
  panel.add(exit);
  return panel;
}
private class MenuListener implements ActionListener
{
  public void actionPerformed(ActionEvent e)
  {
    if(e.getSource() == menuItemCapToFile)//Capture to File
    {
      JFileChooser saveChooser = new JFileChooser();
      try{
        File f = new File(new File(".").getCanonicalPath());
        saveChooser.setCurrentDirectory(f);}
      catch(IOException ioe){System.out.println(ioe);}
      saveChooser.setFileFilter(new FileNameExtensionFilter("wav", "wav"));
      int returnVal = saveChooser.showSaveDialog(null);
      if(returnVal == JFileChooser.APPROVE_OPTION)
      {
        outFile = saveChooser.getSelectedFile();
        if(!outFile.getPath().toLowerCase().endsWith(".wav"))
        {
          outFile = new File(outFile.getPath() + ".wav");
        }
        if(outFile!=null)
```

```
    {
      play.setEnabled(true);
      exit.setEnabled(true);
      srcSelected = 0;
    }
    else
    {
      JOptionPane.showMessageDialog(null, "No File Selected");
    }
  }
  else if(returnVal == JFileChooser.CANCEL_OPTION)
  {
  }
}
else if(e.getSource() == menuItemProcessRealTime)//Real time process
{
  play.setEnabled(true);
  exit.setEnabled(true);
  srcSelected = 1;
}
else if(e.getSource() == menuItemProcessFromFile)//Process from a file
{
  JFileChooser chooser = new JFileChooser();
  try{
    File f = new File(new File(".").getCanonicalPath());
```

```
      chooser.setCurrentDirectory(f);}
    catch(IOException ioe){System.out.println(ioe);}
    chooser.setFileFilter(new FileNameExtensionFilter("wav", "wav"));
    int returnVal = chooser.showOpenDialog(null);
    if(returnVal == JFileChooser.APPROVE_OPTION)
    {
      inFile = chooser.getSelectedFile();
      if(inFile!=null)
      {
        play.setEnabled(true);
        exit.setEnabled(true);
        srcSelected = 2;
      }
      else
      {
        JOptionPane.showMessageDialog(null, "File Did Not Load");
      }
    }
    else if(returnVal == JFileChooser.CANCEL_OPTION)
    {
    }
   }
  }
}
private class ButtonListener implements ActionListener{
```

```
  public void actionPerformed(ActionEvent e){
    JOptionPane pane = new JOptionPane();
    if (e.getSource () == play)
    {
      if(srcSelected == 2){processFromFile();}
      else if(srcSelected == 1)
      {
        float time = Float.parseFloat((timeSpin.getValue()).toString());
        processRealTime(time);
      }
      else if(srcSelected == 0)
      {
        float time = Float.parseFloat((timeSpin.getValue()).toString());
        processRealTimeCapToFile(time);
      }
    }
    else
    {
      System.exit(0);
    }
  }
}
private void processFromFile()
{
  try
```

```
{
  AudioInputStream ais = AudioSystem.getAudioInputStream(inFile);  //reading from here
  AudioFormat format = ais.getFormat();
  Long frameLength = ais.getFrameLength();
  Float fps = format.getFrameRate();
  if(fps == 44100f){
    hzCombo.setSelectedIndex(0);}
  else if(fps == 22050f){
    hzCombo.setSelectedIndex(1);}
  else if(fps == 11025f){
    hzCombo.setSelectedIndex(2);}
  else if(fps == 8000f){
    hzCombo.setSelectedIndex(3);}
  prepareVariables();
  DataLine.Info sourceInfo=new DataLine.Info(SourceDataLine.class,format);
  sourceLine=(SourceDataLine)AudioSystem.getLine(sourceInfo);
  sourceLine.open(format,16384);
  sourceLine.start();              //writing to here
  int k,m;
  writeOffset = readOffset-writeDelay*windowSize;
  if(writeOffset<0){writeOffset+=bufferSize;}
  if(index==2 || index==4) //if pitch shift read first line new data
  {
    ais.read(chunk,0,chunkSize);
    for(int j=0;j<windowSize;j++)
```

```
  {
    k=2*j;
    inBuffer[j+readOffset]=(chunk[k]&255)|(chunk[k+1]<<8);
  }
}
for(int i=0;i<(2*frameLength/chunkSize);i++)//Start of processing loop
{
  ais.read(chunk,0,chunkSize);  //read new data for from File processing
  for(int j=0;j<windowSize;j++)
  {
    k=2*j;
    inBuffer[j+readOffset]=(chunk[k]&255)|(chunk[k+1]<<8);
  }
  if(index==0)processEcho(readOffset);
  if(index==1)processFlange(readOffset);
  if(index==2)processPitchShift(writeOffset);
  if(index==3)processInPlace(readOffset);
  if(index==4)processRobotVoice(writeOffset);
  for(int j=0;j<windowSize;j++)  //write data
  {
    k=2*j;
    m=j+writeOffset;
    chunk[k]=(byte)(((int)outBuffer[m])&255);
    chunk[k+1]=(byte)(((int)outBuffer[m])>>8);
    if(index==2 || index==4)outBuffer[m]=0f; //if pitch Shift
```

```
      }
      sourceLine.write(chunk,0,chunkSize);
      writeOffset=(writeOffset+windowSize)%bufferSize;
      readOffset=(readOffset+windowSize)%bufferSize;
    }
    sourceLine.stop();
    sourceLine.close();
  }
  catch(IOException ioe){System.out.println(ioe);}
  catch(UnsupportedAudioFileException uafe){System.out.println(uafe);}
  catch(LineUnavailableException lue){System.out.println(lue);}
}
private void processRealTime(float time)
{
  try
  {
    prepareVariables();
    AudioFormat fmt = new AudioFormat(fps,16,1,true,false);
    DataLine.Info targetInfo=new DataLine.Info(TargetDataLine.class,fmt);
    DataLine.Info sourceInfo=new DataLine.Info(SourceDataLine.class,fmt);
    targetLine=(TargetDataLine)AudioSystem.getLine(targetInfo);
    sourceLine=(SourceDataLine)AudioSystem.getLine(sourceInfo);
    targetLine.open(fmt,16384);
    sourceLine.open(fmt,16384);
    targetLine.start();              //reading from here
```

```
sourceLine.start();               //writing to here
int k,m;
writeOffset=readOffset-writeDelay*windowSize;
if(writeOffset<0){writeOffset+=bufferSize;}
if(index==2 || index==4) //if pitch shift read first line new data
{
  targetLine.read(chunk,0,chunkSize);
  for(int j=0;j<windowSize;j++)
  {
    k=2*j;
    inBuffer[j+readOffset]=(chunk[k]&255)|(chunk[k+1]<<8);
  }
}
for(int i=0;i<(time*fps)/windowSize;i++)  //Start of processing loop
{
  targetLine.read(chunk,0,chunkSize); //read new data for Real Time processing
  for(int j=0;j<windowSize;j++)
  {
    k=2*j;
    inBuffer[j+readOffset]=(chunk[k]&255)|(chunk[k+1]<<8);
  }
  if(index==0)processEcho(readOffset);
  if(index==1)processFlange(readOffset);
  if(index==2)processPitchShift(writeOffset);
  if(index==3)processInPlace(readOffset);
```

```
    if(index==4)processRobotVoice(writeOffset);
    if(index ==2 && harmonizer.isSelected()==true)processHarmonizer(readOffset);
    for(int j=0;j<windowSize;j++)  //write data
    {
      k=2*j;
      m=j+writeOffset;
      chunk[k]=(byte)(((int)outBuffer[m])&255);
      chunk[k+1]=(byte)(((int)outBuffer[m])>>8);
      if(index==2 || index==4)outBuffer[m]=0f; //for pitch Shift
    }
    sourceLine.write(chunk,0,chunkSize);
    writeOffset=(writeOffset+windowSize)%bufferSize;
    readOffset=(readOffset+windowSize)%bufferSize;
   }
   targetLine.stop();
   sourceLine.stop();
   targetLine.close();
   sourceLine.close();
  }
  catch(LineUnavailableException lue){System.out.println(lue);}
}
private void processRealTimeCapToFile(float time)
{
  try
  {
```

```
prepareVariables();
AudioFormat fmt = new AudioFormat(fps,16,1,true,false);
DataLine.Info targetInfo=new DataLine.Info(TargetDataLine.class,fmt);
targetLine=(TargetDataLine)AudioSystem.getLine(targetInfo);
targetLine.open(fmt,16384);
targetLine.start();                //reading from here
int k,m;
int frameLength = ((int)((time*fps)/2048+.5f))*2048;
byte[] buffer = new byte[frameLength*2];
for(int i=0;i<(frameLength*2/2048);i++)  //Start of processing loop
{
  targetLine.read(buffer, i*2048, 2048); //read new data for Real Time processing
  processInPlace(readOffset);
  readOffset=(readOffset+windowSize)%bufferSize;
  for(int j=0;j<windowSize;j++)  //write data
  {
    k=2*j;
    m=j+writeOffset;
    chunk[k]=(byte)(((int)outBuffer[m])&255);
    chunk[k+1]=(byte)(((int)outBuffer[m])>>8);
    if(index==2 || index==4)outBuffer[m]=0f; //for pitch Shift
  }
  writeOffset=(writeOffset+windowSize)%bufferSize;
}
targetLine.stop();
```

```
    targetLine.close();
    ByteArrayInputStream is = new ByteArrayInputStream(buffer);
    AudioInputStream ais = new AudioInputStream(is,fmt,frameLength);
    FileOutputStream fos = new FileOutputStream(outFile);
    AudioSystem.write(ais,AudioFileFormat.Type.WAVE,fos);
    fos.close();
   }
   catch(IOException ioe){System.out.println(ioe);}
   catch(LineUnavailableException lue){System.out.println(lue);}
}
private void processEcho(int readOffset)
{
   for(int j=0;j<windowSize;j++)
   {
    int placer = j + readOffset;
    outBuffer[placer] = inBuffer[placer];
    float factor = echoDepth;
    for(int i=1;i<=echoRepeat;i++)
    {
     int echoPlacer= (int)(placer-fps*i*echoDelay);
     if(echoPlacer<0) echoPlacer += bufferSize;
     outBuffer[placer] += factor * inBuffer[echoPlacer];
     factor *= echoDepth;
    }
   }
```

```
}
private void processFlange(int readOffset)
{
  for(int j=0;j<windowSize;j++)
  {
    int placer = j + readOffset;
    int flangePlacer= (int)(placer-flangeDelay);
    if(flangePlacer<0) flangePlacer += bufferSize;
    outBuffer[placer]= inBuffer[placer]-flangeDepth*inBuffer[flangePlacer];
    frameCount++;
    if((frameCount % frameSkip) == (frameSkip - 1))
    {
      if(up)
      {
        flangeDelay++;
        if(flangeDelay>=flangeMaxFrames) up=false;
      }
      else
      {
        flangeDelay--;
        if(flangeDelay<=flangeMinFrames) up=true;
      }
    }
  }
}
```

```
 private void processPitchShift(int readOffset)
 {
  for(int h=0;h<overlap;h++)
  {
   float[][] fft=FFTransform.realDataForwardXform(inBuffer,readOffset+h*hopA,windowSize
                ,bufferSize,hannWindowA);
   for(int j=0;j<windowSize/2;j++)
   {
    real=fft[0][j];
    imag=fft[1][j];
    mag=(float)Math.sqrt(real*real+imag*imag);
    phiA=getPhase(real,imag);
    wBin=(2*pi*j*hopA)/windowSize;
    deltaPhi=phiA-prevPhiA[j]-wBin;
    deltaPhiWrap=wrapPhase(deltaPhi);
    wTrue=(wBin+deltaPhiWrap)/hopA;
    phiS=wrapPhase(prevPhiS[j]+hopS*wTrue);
//       fft[0][j]=4*mag;
//       fft[1][j]=0;
    fft[0][j]=mag*(float)Math.cos(phiS);
    fft[1][j]=mag*(float)Math.sin(phiS);
    if(j>0) fft[0][windowSize-j]=fft[0][j];
    if(j>0) fft[1][windowSize-j]=-fft[1][j];
   }
   FFTransform.realDataInverseXformOverlapAdd(fft,outBuffer,readOffset+h*hopA,windowSize
```

```
          ,bufferSize,hannWindowS,beta);
  }
}
private void processHarmonizer(int readOffset)
{
  for(int j=0;j<windowSize;j++)
  {
    outBuffer[j+readOffset]=inBuffer[j+readOffset];
  }
}
private void processRobotVoice(int readOffset)
{
  for(int h=0;h<overlap;h++)
  {
    float[][] fft=FFTransform.realDataForwardXform(inBuffer,readOffset+h*hopA,windowSize
                  ,bufferSize,hannWindowA);
    for(int j=0;j<windowSize/2;j++)
    {
      real=fft[0][j];
      imag=fft[1][j];
      mag=(float)Math.sqrt(real*real+imag*imag);
      phiA=getPhase(real,imag);
      wBin=(2*pi*j*hopA)/windowSize;
      deltaPhi=phiA-prevPhiA[j]-wBin;
      deltaPhiWrap=wrapPhase(deltaPhi);
```

```
        wTrue=(wBin+deltaPhiWrap)/hopA;
        phiS=wrapPhase(prevPhiS[j]+hopS*wTrue);
        fft[0][j]=4*mag;
        fft[1][j]=0;
//        fft[0][j]=mag*(float)Math.cos(phiS);
//        fft[1][j]=mag*(float)Math.sin(phiS);
        if(j>0) fft[0][windowSize-j]=fft[0][j];
        if(j>0) fft[1][windowSize-j]=-fft[1][j];
      }
      FFTransform.realDataInverseXformOverlapAdd(fft,outBuffer,readOffset+h*hopA,windowSize
            ,bufferSize,hannWindowS,beta);
    }
  }
  private void processInPlace(int readOffset)
  {
    for(int j=0;j<windowSize;j++)
    {
      outBuffer[j+readOffset]=inBuffer[j+readOffset];
    }
  }
  private void prepareVariables()
  {
    fps = Float.parseFloat((hzCombo.getSelectedItem().toString()));
    windowSize = Integer.parseInt((windowSizeCombo.getSelectedItem().toString()));
    index = processType();
```

```
if(index==0) //variables for echo
{
  echoDelay = Float.parseFloat((echoDelaySpin.getValue()).toString())/1000;
  echoDepth = Float.parseFloat((echoDepthSpin.getValue()).toString());
  echoRepeat = Integer.parseInt((echoRepeatSpin.getValue()).toString());
  numWindows=(int)(fps*echoRepeat*echoDelay/windowSize+2);
}
else if(index==1) //variables for flanging
{
  flangeMinDelay = (Float.parseFloat((flangeMinDelaySpin.getValue()).toString()))/10000;
  flangeMaxDelay = Float.parseFloat((flangeMaxDelaySpin.getValue()).toString())/1000;
  flangeDepth = Float.parseFloat((flangeDepthSpin.getValue()).toString());
  flangePeriod = Float.parseFloat((flangePeriodSpin.getValue()).toString())/1000;
  numWindows = (int)(fps*flangeMaxDelay/windowSize+2);
  flangeMinFrames = (int)(flangeMinDelay*fps);
  flangeMaxFrames = (int)(flangeMaxDelay*fps);
  flangeDelay = flangeMinFrames;
  frameSkip = (int)(flangePeriod/((flangeMaxDelay-flangeMinDelay)*2));
}
else if(index==2 || index==4) //variables for pitch shift
{
  if(index==2)beta = Float.parseFloat((betaPitchSpin.getValue()).toString());
  if(index==4)beta = Float.parseFloat((betaRobotSpin.getValue()).toString());
  hopA = windowSize/overlap;
  hopS = (int)(hopA*beta+.5f);
```

```
 scaleA = (float)Math.sqrt(windowSize/(2f*hopA));
 scaleS = (float)Math.sqrt(windowSize/(2f*hopS));
 hannWindowA = new float[windowSize];
 hannWindowS = new float[windowSize];
 prevPhiA = new float[windowSize];
 prevPhiS = new float[windowSize];
 numWindows = 5;
 writeDelay = 1;
 float w;
 for(int i=0;i<hannWindowA.length;i++)
 {
  w = 0.5f*(float)(1-Math.cos((2*pi*i)/(windowSize-1)));
  hannWindowA[i] = w/scaleA;
  hannWindowS[i] = w/scaleS;
 }
}
else //variables for no processing
{
 numWindows = 5;
}
bufferSize = windowSize*numWindows;
inBuffer = new float[bufferSize];
outBuffer = new float[bufferSize];
chunkSize = 2*windowSize;
chunk = new byte[chunkSize];
```

```
  readOffset = 0;
  writeOffset = 0;
}
private float getPhase(float a,float b)
{
  float phase;
  if(a==0f)
  {
    if(b>=0f) phase=pi/2;
    else phase=-pi/2;
  }
  else
  {
     phase=(float)Math.atan(b/a);
     if(a<0f&&b>0f) phase+=pi;
     else if(a<0f&&b<0f) phase-=pi;
  }
  return phase;
}
private float wrapPhase(float theta)
{
  if(theta>0){  while(theta>pi) theta-=2*pi;}
  else{ while(theta<-pi) theta+=2*pi;}
  return theta;
}
```

```
  protected int processType()
  {
    int index = tabbedPane.getSelectedIndex();
    return index;
  }
  public static void main(String[] args)
  {
    new LiveSoundUtility();
  }
}
```