Austin Orth

Dr. Chang

Multimedia Software Engineering

November 29, 2016

Breath Component

Final Report

# Project Description

The Breath component is a multipurpose microphone-based audio energy measurement component designed for integration into the Slow Intelligence System server. Using a decibel threshold, the component can detect when a patient breaths and record the time and decibel level of the breath as a reading. The component also sends an alert message to the server and uploader with the time and level of last breath when a breath has not been detected for ten seconds.

When started, the component allows the user to set a refresh rate, start date, and end date for audio measurement using the standard SIS GUI. Then, once the user clicks the activate button, a second GUI is launched which allows the user to select any audio recording driver associated with a device installed on the current machine. This additional GUI also allows the user to set a listening threshold corresponding to the decibel level of breaths that the user wishes to be detected.

Once all parameters have been initialized by the user, the component will begin to listen for breath on the selected recording device. Each breath that crosses the threshold set by the user will display as a breath reading in the text-based reporter along with the decibel level of that breath. If the patient’s breath does not pass the threshold for ten seconds, an alert will be sent to the server and the uploader to notify that the patient may be experiencing breathing difficulty or irregularity.

###Extra Deeds###

1. I changed the original text entry for listening threshold to a slider, allowing the threshold to be easily set by inexperienced users.
2. Added universal support for audio drivers using TarsosDSP.
3. Added visual monitor that shows the set threshold and the loudest breath heard.
4. Set lines being drawn for decibel levels to turn green when passing the threshold and yellow when under the threshold, signifying when patient is breathing and not breathing respectively.

# Component Diagram



## Interaction Key

* m1 = breath component initialization and settings message
* m2 = Parameters with Activate, Deactivate, or Kill command
* m3 = initialized shared variables
* m4 = initialize GUI
* m5 = user settings
* m6 = shared variables modified with current breath data
* m7 = reading message
* m8 = alert message
* m9 = database query

# Example Scenario

## Description

This scenario illustrates with helpful instructions what a user would do to monitor and measure breath using the Breath component. Each step corresponds to screenshots below.

## Steps

1. Navigate to the “Scripts” folder within the SISv5 directory and double-click “runserver.bat” to launch an SIS server locally.
2. A console window will then appear showing the status of the server.
3. Double-click “runInitializer.bat”.
4. Upon checking the server console window, you can see that the Breath component has been initialized.
5. Navigate inside the “runIndividualComp” directory and double-click “runGUI.bat”.
6. This will launch the standard SIS GUI container window, which will display GUIs for each component we run.
7. Double-click “runUploader.bat”.
8. This will launch the default SIS Uploader component, which I have modified to support Breath component data. By looking at the corresponding console window, you can see that Breath data is included in the initial database update.
9. The Uploader GUI element also displays correctly within the default SIS GUI container window.
10. Double-click “runBreathNoCompile.bat” to run the Breath component. (This saves time compared to the original batch file, which recompiles every time it is run.)
11. Now if you check the SIS GUI window, you will see an additional GUI module has appeared, allowing you to set the refresh rate, start date, and end date for the Breath component, as well as activate it.
12. Once you have set the parameters, click the button labeled “Inactive” on the SIS GUI module for Breath to activate an instance of the component.
13. A new window will now appear with a Breath-specific GUI.
14. Select the audio driver you wish to use for Breath monitoring from the displayed list. The Breath component will then begin monitoring using the device associated with that driver. Input from the device can be seen on the visual monitor on the bottom half of the window.
15. Breath levels may vary from person to person. Adjust the decibel threshold slider to your preference. You can use the visual monitor to guide your decision. The new level you select is indicated by the yellow bar, as well as the text outputted in the text box.
16. To check if the upload feature is working, you can set the threshold too high on purpose as shown here.
17. Check the Uploader window after ten seconds. You will see that the Uploader has sent an alert to the database.

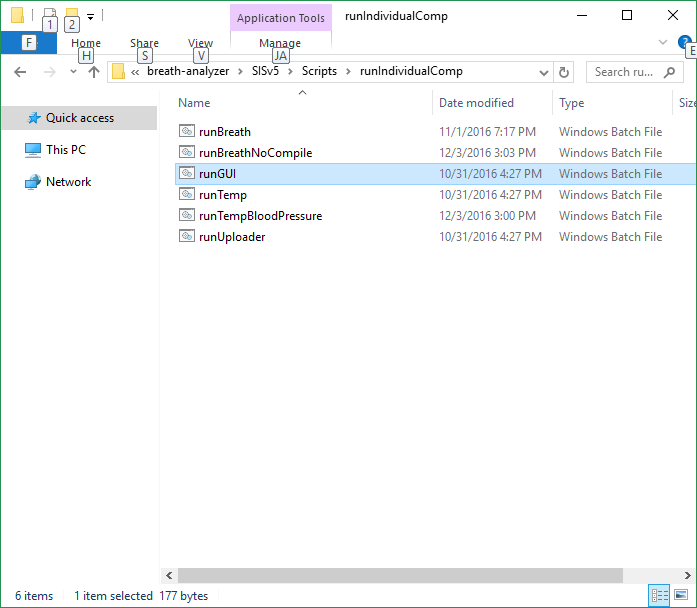
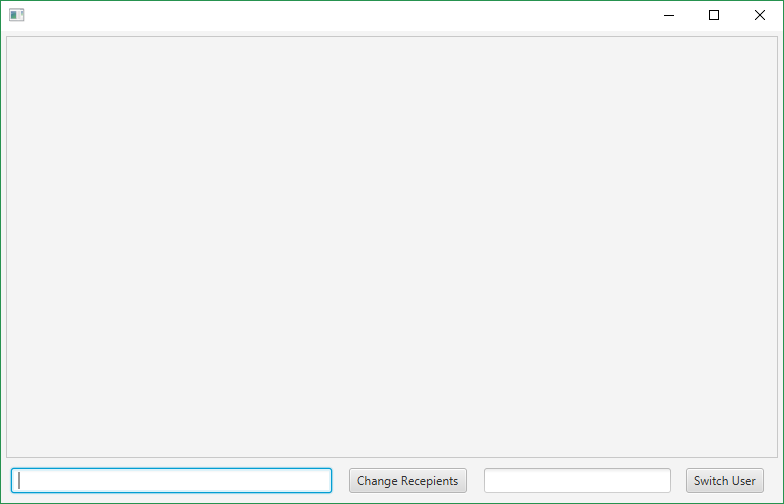
## Demo Screenshots

4

3

2

1

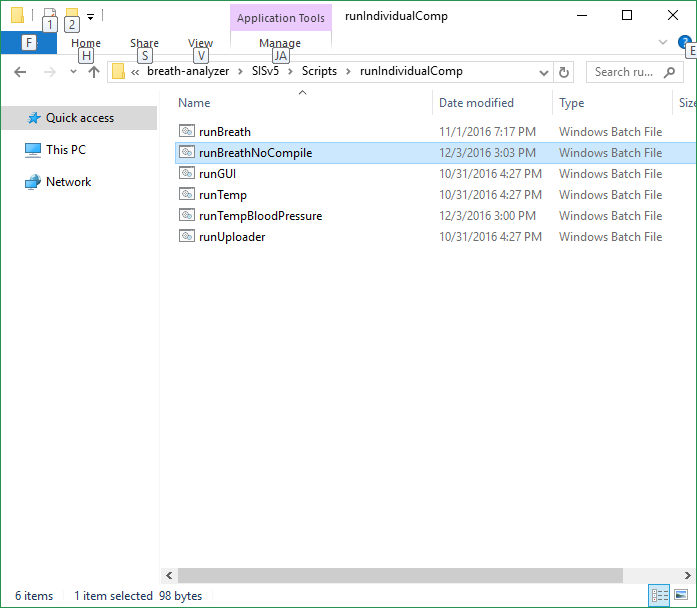
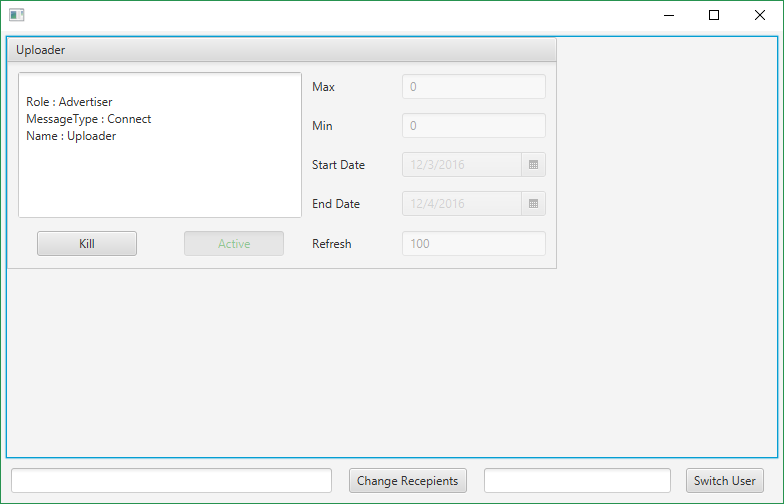
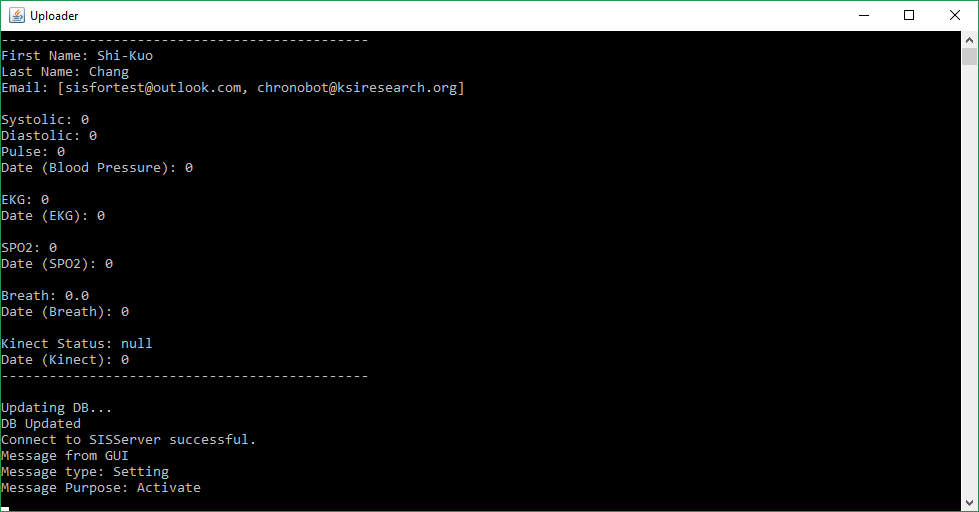
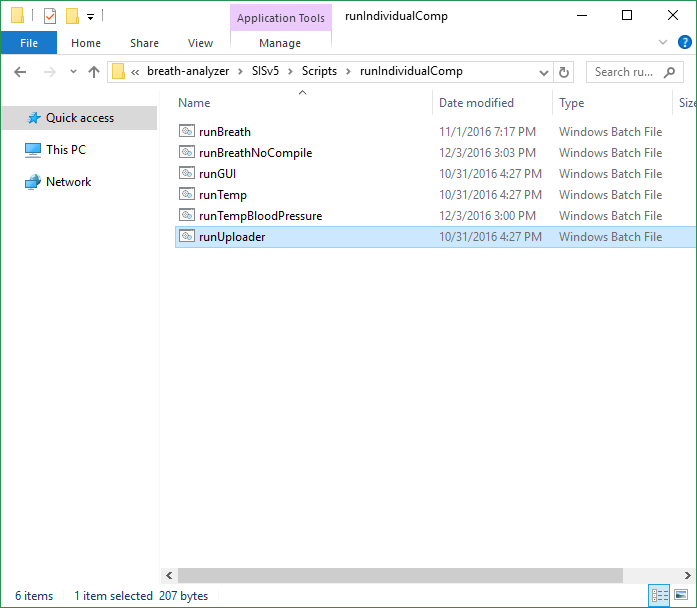


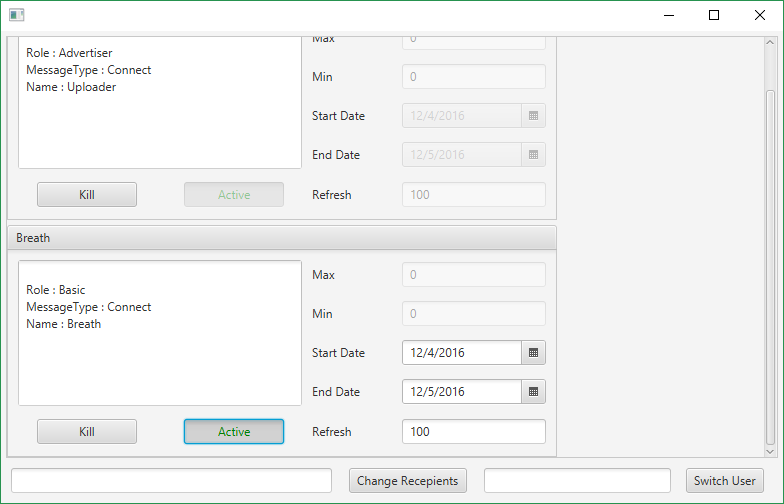
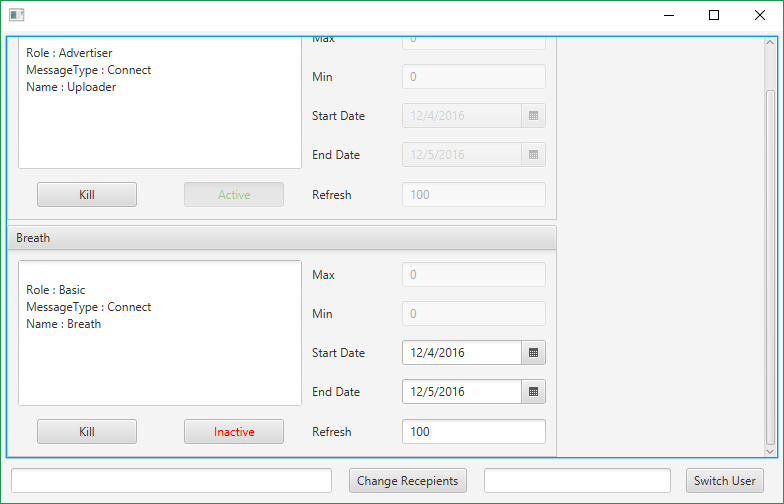
6

5

8

7



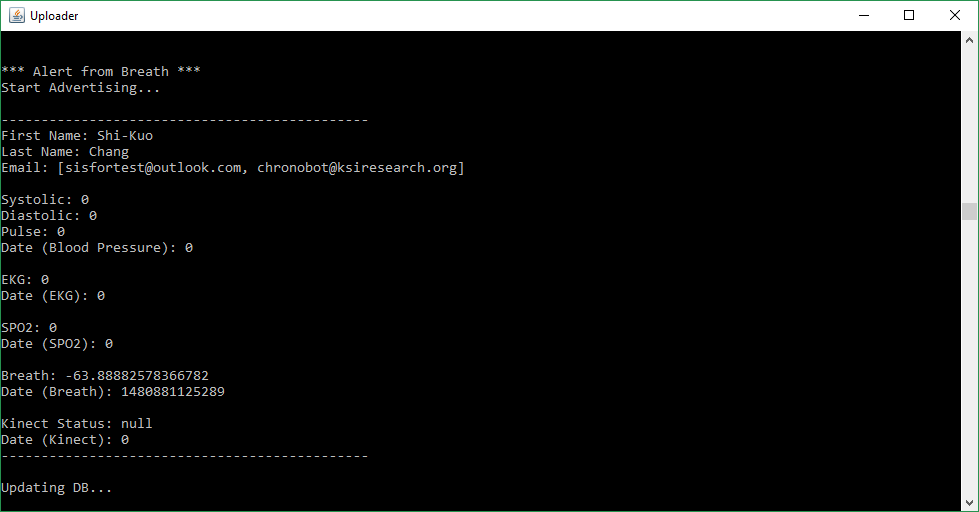
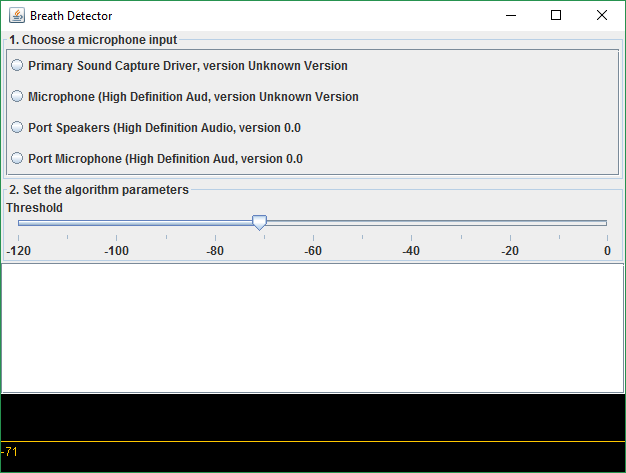
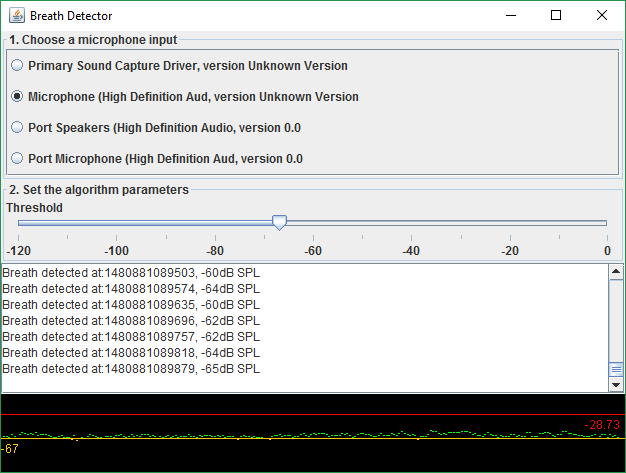
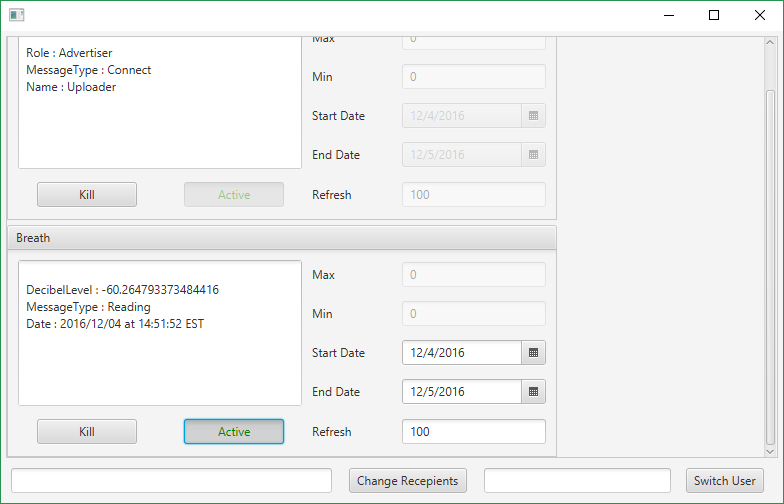
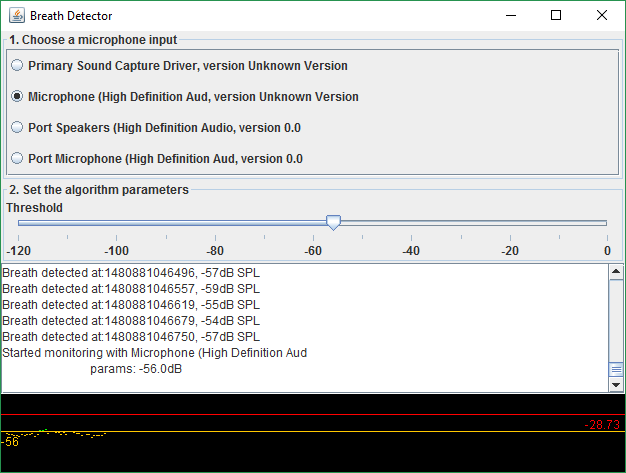


11

12

9

10



16

15

17

14

13

# Appendix

## Third Party Libraries Used

* TarsosDSP
  + “TarsosDSP is a Java library for audio processing. Its aim is to provide an easy-to-use interface to practical music processing algorithms implemented, as simply as possible, in pure Java and without any other external dependencies. The library tries to hit the sweet spot between being capable enough to get real tasks done but compact and simple enough to serve as a demonstration on how DSP algorithms works. TarsosDSP features an implementation of a percussion onset detector and a number of pitch detection algorithms: YIN, the Mcleod Pitch method and a “Dynamic Wavelet Algorithm Pitch Tracking” algorithm. Also included is a Goertzel DTMF decoding algorithm, a time stretch algorithm (WSOLA), resampling, filters, simple synthesis, some audio effects, and a pitch shifting algorithm.”
  + https://github.com/JorenSix/TarsosDSP

## Source Code

BreathDetector.java

|  |
| --- |
| import java.awt.BorderLayout;  import java.awt.Color;  import java.awt.Dimension;  import java.awt.Graphics;  import java.beans.PropertyChangeEvent;  import java.beans.PropertyChangeListener;  import java.lang.reflect.InvocationTargetException;  import java.util.ArrayList;  import java.util.List;  import javax.sound.sampled.AudioFormat;  import javax.sound.sampled.AudioInputStream;  import javax.sound.sampled.Clip;  import javax.sound.sampled.DataLine;  import javax.sound.sampled.LineUnavailableException;  import javax.sound.sampled.Mixer;  import javax.sound.sampled.TargetDataLine;  import javax.sound.sampled.UnsupportedAudioFileException;  import javax.swing.JFrame;  import javax.swing.JLabel;  import javax.swing.JPanel;  import javax.swing.JScrollPane;  import javax.swing.JSlider;  import javax.swing.JTextArea;  import javax.swing.SwingUtilities;  import javax.swing.border.TitledBorder;  import javax.swing.event.ChangeEvent;  import javax.swing.event.ChangeListener;  import be.tarsos.dsp.AudioDispatcher;  import be.tarsos.dsp.AudioEvent;  import be.tarsos.dsp.AudioProcessor;  import be.tarsos.dsp.SilenceDetector;  import be.tarsos.dsp.io.jvm.JVMAudioInputStream;  public class BreathDetector extends JFrame implements AudioProcessor {  private static final long serialVersionUID = 3501426880288136245L;  private final JTextArea textArea;  ArrayList<Clip> clipList;  public static int counter;  public static double breathLevel;  public static boolean stopped = true;  public static long breathTime;  public static JFrame frame;  double threshold;  AudioDispatcher dispatcher;  Mixer currentMixer;  private final GaphPanel graphPanel;  SilenceDetector silenceDetector;  public BreathDetector() {  this.setLayout(new BorderLayout());  this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  this.setTitle("Breath Detector");  this.threshold = SilenceDetector.DEFAULT\_SILENCE\_THRESHOLD;  JPanel inputPanel = new InputPanel();  //add(inputPanel);  inputPanel.addPropertyChangeListener("mixer",  new PropertyChangeListener() {  @Override  public void propertyChange(PropertyChangeEvent arg0) {  try {  setNewMixer((Mixer) arg0.getNewValue());  } catch (LineUnavailableException e) {  // TODO Auto-generated catch block  e.printStackTrace();  } catch (UnsupportedAudioFileException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  }  });  JSlider thresholdSlider = initialzeThresholdSlider();  JPanel params = new JPanel(new BorderLayout());  params.setBorder(new TitledBorder("2. Set the algorithm parameters"));  JLabel label = new JLabel("Threshold");  label.setToolTipText("Energy level when breath is counted (dB SPL).");  params.add(label,BorderLayout.NORTH);  params.add(thresholdSlider,BorderLayout.CENTER);  JPanel inputAndParamsPanel = new JPanel(new BorderLayout());  inputAndParamsPanel.add(inputPanel,BorderLayout.NORTH);  inputAndParamsPanel.add(params,BorderLayout.SOUTH);  JPanel panelWithTextArea = new JPanel(new BorderLayout());  textArea = new JTextArea(8,30);  textArea.setEditable(false);  panelWithTextArea.add(inputAndParamsPanel,BorderLayout.NORTH);  panelWithTextArea.add(new JScrollPane(textArea),BorderLayout.CENTER);  add(panelWithTextArea,BorderLayout.NORTH);  graphPanel = new GaphPanel(threshold);  graphPanel.setSize(80,100);  add(graphPanel,BorderLayout.CENTER);  }  private static class GaphPanel extends JPanel{  private static final long serialVersionUID = 5969781241442094359L;  private double threshold;  private double maxLevel = -1000;  private long currentModulo = System.currentTimeMillis()/15000;  private List<Double> levels;  private List<Long> startTimes;  public GaphPanel(double defaultThreshold){  setThresholdLevel(defaultThreshold);  levels = new ArrayList<Double>();  startTimes=new ArrayList<Long>();  setMinimumSize(new Dimension(80,60));  }  private void setMaxLevel(double newMaxLevel){  if(newMaxLevel> maxLevel){  maxLevel=newMaxLevel;  }  }  public void setThresholdLevel(double newThreshold){  threshold=newThreshold;  repaint();  }  public void addDataPoint(double level,long ms){  levels.add(level);  startTimes.add(ms);  setMaxLevel(level);  repaint();  }  public void paintComponent(Graphics g) {  super.paintComponent(g); //paint background  g.setColor(Color.BLACK);  g.fillRect(0, 0,getWidth(), getHeight());  if(System.currentTimeMillis()/15000 > currentModulo){  currentModulo = System.currentTimeMillis()/15000;  levels.clear();  startTimes.clear();  }  for(int i =0 ; i < levels.size();i++){  g.setColor( levels.get(i) > threshold ? Color.GREEN:Color.ORANGE );  int x = msToXCoordinate(startTimes.get(i));  int y = levelToYCoordinate(levels.get(i));  g.drawLine(x, y, x+1, y);  }  int thresholdYCoordinate = levelToYCoordinate(threshold);  g.setColor(Color.ORANGE);  g.drawLine(0, thresholdYCoordinate, getWidth(),thresholdYCoordinate);  g.drawString(String.valueOf((int)threshold), 0, thresholdYCoordinate + 15);  int maxYCoordinate = levelToYCoordinate(maxLevel);  g.setColor(Color.RED);  g.drawLine(0, maxYCoordinate, getWidth(),maxYCoordinate);  g.drawString(String.valueOf(((int)(maxLevel\*100))/100.0), getWidth() - 40, maxYCoordinate + 15);  }  private int levelToYCoordinate(double level){  int inPixels = (int)((120 + level) / 120 \* (getHeight()-1));  int yCoordinate = getHeight() - inPixels;  return yCoordinate;  }  private int msToXCoordinate(long ms){  return (int) ((ms % 15000)/15000.0 \* getWidth());  }  }  private JSlider initialzeThresholdSlider() {  JSlider thresholdSlider = new JSlider(-120,0);  thresholdSlider.setValue((int)threshold);  thresholdSlider.setPaintLabels(true);  thresholdSlider.setPaintTicks(true);  thresholdSlider.setMajorTickSpacing(20);  thresholdSlider.setMinorTickSpacing(10);  thresholdSlider.addChangeListener(new ChangeListener(){  @Override  public void stateChanged(ChangeEvent e) {  JSlider source = (JSlider) e.getSource();  threshold = source.getValue();  graphPanel.setThresholdLevel(threshold);  if (!source.getValueIsAdjusting()) {  try {  setNewMixer(currentMixer);  } catch (LineUnavailableException e1) {  // TODO Auto-generated catch block  e1.printStackTrace();  } catch (UnsupportedAudioFileException e1) {  // TODO Auto-generated catch block  e1.printStackTrace();  }  }  }  });  return thresholdSlider;  }  private void setNewMixer(Mixer mixer) throws LineUnavailableException,  UnsupportedAudioFileException {  if(dispatcher!= null){  dispatcher.stop();  }  currentMixer = mixer;  float sampleRate = 44100;  int bufferSize = 512;  int overlap = 0;  textArea.append("Started monitoring with " + Shared.toLocalString(mixer.getMixerInfo().getName()) + "\n\tparams: " + threshold + "dB\n");  final AudioFormat format = new AudioFormat(sampleRate, 16, 1, true,  true);  final DataLine.Info dataLineInfo = new DataLine.Info(  TargetDataLine.class, format);  TargetDataLine line;  line = (TargetDataLine) mixer.getLine(dataLineInfo);  final int numberOfSamples = bufferSize;  line.open(format, numberOfSamples);  line.start();  final AudioInputStream stream = new AudioInputStream(line);  JVMAudioInputStream audioStream = new JVMAudioInputStream(stream);  // create a new dispatcher  dispatcher = new AudioDispatcher(audioStream, bufferSize,  overlap);  // add a processor, handle percussion event.  silenceDetector = new SilenceDetector(threshold,false);  dispatcher.addAudioProcessor(silenceDetector);  dispatcher.addAudioProcessor(this);  // run the dispatcher (on a new thread).  new Thread(dispatcher,"Audio dispatching").start();  }  public static void main(String... strings) throws InterruptedException,  InvocationTargetException {  SwingUtilities.invokeAndWait(new Runnable() {  @Override  public void run() {  frame = new BreathDetector();  frame.pack();  frame.setSize(640,480);  frame.setVisible(true);  stopped = false;  }  });  }  @Override  public boolean process(AudioEvent audioEvent) {  handleSound();  return true;  }  private void handleSound(){  if(silenceDetector.currentSPL() > threshold){  breathLevel = (double)(silenceDetector.currentSPL());  breathTime = System.currentTimeMillis();  textArea.append("Breath detected at:" + System.currentTimeMillis() + ", " + (int)(silenceDetector.currentSPL()) + "dB SPL\n");  textArea.setCaretPosition(textArea.getDocument().getLength());  }  graphPanel.addDataPoint(silenceDetector.currentSPL(), System.currentTimeMillis());  }  @Override  public void processingFinished() {  }  } |

CustomControl.java

|  |
| --- |
| import java.io.IOException;  import java.net.Socket;  import java.net.UnknownHostException;  import java.time.Instant;  import java.time.LocalDate;  import java.time.ZoneId;  import java.util.Date;  import javafx.beans.property.ObjectProperty;  import javafx.beans.property.StringProperty;  import javafx.beans.value.ChangeListener;  import javafx.beans.value.ObservableValue;  import javafx.event.EventHandler;  import javafx.fxml.FXML;  import javafx.fxml.FXMLLoader;  import javafx.scene.CacheHint;  import javafx.scene.control.Button;  import javafx.scene.control.DatePicker;  import javafx.scene.control.TextArea;  import javafx.scene.control.TextField;  import javafx.scene.control.TitledPane;  import javafx.scene.control.ToggleButton;  import javafx.scene.input.InputMethodEvent;  import javafx.scene.input.MouseButton;  import javafx.scene.input.MouseEvent;  import javafx.scene.layout.VBox;  import javafx.scene.paint.Color;  import javafx.scene.text.Font;  public class CustomControl extends VBox {  Socket universal;  static int port = 53217;  MsgEncoder encoder;  //private final String SCOPE = "SIS.Scope1";  @FXML  private TitledPane title;  @FXML  private TextArea console;  @FXML  private TextField max;  @FXML  private TextField min;  @FXML  private DatePicker startDate;  @FXML  private DatePicker endDate;  @FXML  private TextField refreshRate;  @FXML  private Button kill;  @FXML  private ToggleButton active;  public CustomControl() {  // TODO Auto-generated constructor stub  FXMLLoader fxmlLoader = new FXMLLoader(getClass().getResource(  "CustomControl.fxml"));  fxmlLoader.setRoot(this);  fxmlLoader.setController(this);  try {  fxmlLoader.load();  //setCache(true);  //setCacheShape(true);  //setCacheHint(CacheHint.SPEED);  setMin(0 + "");  setMax(0 + "");  setStartDate(LocalDate.now());  LocalDate tomorrow = LocalDate.now().withDayOfMonth(LocalDate.now().getDayOfMonth() + 1);  setEndDate(tomorrow);  setRefreshRate(100 + "");  min.textProperty().addListener(new ChangeListener<String>() {  @Override  public void changed(  ObservableValue<? extends String> observable,  String oldValue, String newValue) {  try {  int mi = Integer.parseInt(newValue);  int ma = Integer.parseInt(max.textProperty().get());  if (mi > ma) {  max.setText(mi + "");  }  min.setText(mi + "");  } catch (NumberFormatException e) {  min.setText(oldValue);  }  }  });  max.textProperty().addListener(new ChangeListener<String>() {  @Override  public void changed(  ObservableValue<? extends String> observable,  String oldValue, String newValue) {  try {  int ma = Integer.parseInt(newValue);  int mi = Integer.parseInt(min.textProperty().get());  if (ma < mi) {  min.setText(ma + "");  }  max.setText(ma + "");  } catch (NumberFormatException e) {  max.setText(oldValue);  }  }  });  startDate.valueProperty().addListener(  new ChangeListener<LocalDate>() {  @Override  public void changed(  ObservableValue<? extends LocalDate> observable,  LocalDate oldValue, LocalDate newValue) {  // TODO Auto-generated method stub  LocalDate end = endDate.valueProperty().get();  if (newValue.isBefore(LocalDate.now())) {  startDate.setValue(oldValue);  }  if (newValue.isAfter(end) || newValue.isEqual(end)) {  LocalDate tomorrow = newValue.withDayOfMonth(newValue.getDayOfMonth() + 1);  endDate.setValue(tomorrow);  }  }  });  endDate.valueProperty().addListener(  new ChangeListener<LocalDate>() {  @Override  public void changed(  ObservableValue<? extends LocalDate> observable,  LocalDate oldValue, LocalDate newValue) {  // TODO Auto-generated method stub  LocalDate start = startDate.valueProperty().get();  ///if (newValue.isBefore(LocalDate.now())) {  /// endDate.setValue(oldValue);  ///}  if (newValue.isBefore(start) || newValue.isEqual(start)) {  endDate.setValue(oldValue);  }  }  });  refreshRate.textProperty().addListener(  new ChangeListener<String>() {  @Override  public void changed(  ObservableValue<? extends String> observable,  String oldValue, String newValue) {  try {  int val = Integer.parseInt(newValue);  if (val > 100) {  refreshRate.setText(val + "");  } else {  refreshRate.setText(100 + "");  }  } catch (NumberFormatException e) {  refreshRate.setText(oldValue);  }  }  });  // title.setOnMouseClicked(new EventHandler<MouseEvent>() {  //  // @Override  // public void handle(MouseEvent event) {  // // TODO Auto-generated method stub  // MouseButton button = event.getButton();  // if (button == MouseButton.SECONDARY) {  // title.setTextFill(Color.BLACK);  // }  // }  // });  } catch (IOException exception) {  throw new RuntimeException(exception);  }  }  public void handlerAlertResolved(MouseEvent event) {  MouseButton button = event.getButton();  if (button == MouseButton.SECONDARY) {  title.setTextFill(Color.BLACK);  }  }  public void handlerKill(MouseEvent event) {  // if (selected) {  // boot.setText("Alive");  // boot.setTextFill(Color.GREEN);  // active.setDisable(false);  // active.setSelected(true);  // handlerActiveInactive(null);  //  // try {  // if (universal == null) {  // universal = new Socket("127.0.0.1", port);  // }  // if (encoder == null) {  // encoder = new MsgEncoder(universal.getOutputStream());  // }  //  // KeyValueList bootMsg = new KeyValueList();  // bootMsg.putPair("Scope", SCOPE);  // bootMsg.putPair("MessageType", "Boot");  // bootMsg.putPair("Name", getTitle());  //  // encoder.sendMsg(bootMsg);  //  // } catch (UnknownHostException e) {  // // TODO Auto-generated catch block  // e.printStackTrace();  // } catch (IOException e) {  // // TODO Auto-generated catch block  // e.printStackTrace();  // }  try {  if (universal == null) {  universal = new Socket("127.0.0.1", port);  }  if (encoder == null) {  encoder = new MsgEncoder(universal.getOutputStream());  }  KeyValueList kill = new KeyValueList();  kill.putPair("Scope", CreateGUI.SCOPE);  kill.putPair("MessageType", "Setting");  kill.putPair("Sender", CreateGUI.NAME);  kill.putPair("Receiver", getTitle());  kill.putPair("Purpose", "Kill");  encoder.sendMsg(kill);  active.setSelected(false);  handlerActiveInactive(null);  active.setDisable(true);  } catch (UnknownHostException e) {  // TODO Auto-generated catch block  e.printStackTrace();  } catch (IOException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  }  public void handlerActiveInactive(MouseEvent event) {  boolean selected = active.selectedProperty().get();  if (selected) {  active.setText("Active");  active.setTextFill(Color.GREEN);  try {  if (universal == null) {  universal = new Socket("127.0.0.1", port);  }  if (encoder == null) {  encoder = new MsgEncoder(universal.getOutputStream());  }  KeyValueList act = new KeyValueList();  act.putPair("Scope", CreateGUI.SCOPE);  act.putPair("MessageType", "Setting");  act.putPair("Sender", CreateGUI.NAME);  act.putPair("Receiver", getTitle());  act.putPair("Purpose", "Activate");  act.putPair("Max", getMax());  act.putPair("Min", getMin());  LocalDate st = getStartDate();  Instant instantS = st.atStartOfDay()  .atZone(ZoneId.systemDefault()).toInstant();  Date dateS = Date.from(instantS);  LocalDate ed = getEndDate();  Instant instantE = ed.atStartOfDay()  .atZone(ZoneId.systemDefault()).toInstant();  Date dateE = Date.from(instantE);  act.putPair("StartDate", dateS.getTime() + "");  act.putPair("EndDate", dateE.getTime() + "");  act.putPair("RefreshRate", getRefreshRate());  encoder.sendMsg(act);  } catch (UnknownHostException e) {  // TODO Auto-generated catch block  e.printStackTrace();  } catch (IOException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  } else {  active.setText("Inactive");  active.setTextFill(Color.RED);  try {  if (universal == null) {  universal = new Socket("127.0.0.1", port);  }  if (encoder == null) {  encoder = new MsgEncoder(universal.getOutputStream());  }  KeyValueList dact = new KeyValueList();  dact.putPair("Scope", CreateGUI.SCOPE);  dact.putPair("MessageType", "Setting");  dact.putPair("Sender", CreateGUI.NAME);  dact.putPair("Receiver", getTitle());  dact.putPair("Purpose", "Deactivate");  encoder.sendMsg(dact);  } catch (UnknownHostException e) {  // TODO Auto-generated catch block  e.printStackTrace();  } catch (IOException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  }  }  public void setAlert() {  title.setTextFill(Color.RED);  }  public void setTitle(String t) {  titleProperty().set(t);  }  public String getTitle() {  return titleProperty().get();  }  public StringProperty titleProperty() {  return title.textProperty();  }  public String getConsole() {  return consoleProperty().get();  }  public void setConsole(String value) {  consoleProperty().set(value);  }  public void setActive() {  active.setDisable(false);  active.setSelected(true);  active.setText("Active");  active.setTextFill(Color.GREEN);  }  public void setEnable() {  active.setDisable(false);  }  public StringProperty consoleProperty() {  return console.textProperty();  }  public String getMax() {  return maxProperty().get();  }  public void setMax(String value) {  maxProperty().set(value);  }  public StringProperty maxProperty() {  return max.textProperty();  }  public String getMin() {  return minProperty().get();  }  public void setMin(String value) {  minProperty().set(value);  }  public StringProperty minProperty() {  return min.textProperty();  }  public String getRefreshRate() {  return refreshRateProperty().get();  }  public void setRefreshRate(String value) {  refreshRateProperty().set(value);  }  public StringProperty refreshRateProperty() {  return refreshRate.textProperty();  }  public void setStartDate(LocalDate date) {  startDateProperty().set(date);  }  public LocalDate getStartDate() {  return startDateProperty().get();  }  public ObjectProperty<LocalDate> startDateProperty() {  return startDate.valueProperty();  }  public void setEndDate(LocalDate date) {  endDateProperty().set(date);  }  public LocalDate getEndDate() {  return endDateProperty().get();  }  public ObjectProperty<LocalDate> endDateProperty() {  return endDate.valueProperty();  }  public void setSorAItems(){  max.setDisable(true);  min.setDisable(true);  startDate.setDisable(true);  endDate.setDisable(true);  refreshRate.setDisable(true);  active.setSelected(true);  handlerActiveInactive(null);  active.setDisable(true);  }  public void setAudioItems(){  max.setDisable(true);  min.setDisable(true);  startDate.setDisable(false);  endDate.setDisable(false);  refreshRate.setDisable(false);  }  } |

CreateUploader.java

|  |
| --- |
| import java.io.IOException;  import java.net.Socket;  import java.net.UnknownHostException;  import java.time.Instant;  import java.time.LocalDate;  import java.time.ZoneId;  import java.util.Date;  import javafx.beans.property.ObjectProperty;  import javafx.beans.property.StringProperty;  import javafx.beans.value.ChangeListener;  import javafx.beans.value.ObservableValue;  import javafx.event.EventHandler;  import javafx.fxml.FXML;  import javafx.fxml.FXMLLoader;  import javafx.scene.CacheHint;  import javafx.scene.control.Button;  import javafx.scene.control.DatePicker;  import javafx.scene.control.TextArea;  import javafx.scene.control.TextField;  import javafx.scene.control.TitledPane;  import javafx.scene.control.ToggleButton;  import javafx.scene.input.InputMethodEvent;  import javafx.scene.input.MouseButton;  import javafx.scene.input.MouseEvent;  import javafx.scene.layout.VBox;  import javafx.scene.paint.Color;  import javafx.scene.text.Font;  public class CustomControl extends VBox {  Socket universal;  static int port = 53217;  MsgEncoder encoder;  //private final String SCOPE = "SIS.Scope1";  @FXML  private TitledPane title;  @FXML  private TextArea console;  @FXML  private TextField max;  @FXML  private TextField min;  @FXML  private DatePicker startDate;  @FXML  private DatePicker endDate;  @FXML  private TextField refreshRate;  @FXML  private Button kill;  @FXML  private ToggleButton active;  public CustomControl() {  // TODO Auto-generated constructor stub  FXMLLoader fxmlLoader = new FXMLLoader(getClass().getResource(  "CustomControl.fxml"));  fxmlLoader.setRoot(this);  fxmlLoader.setController(this);  try {  fxmlLoader.load();  //setCache(true);  //setCacheShape(true);  //setCacheHint(CacheHint.SPEED);  setMin(0 + "");  setMax(0 + "");  setStartDate(LocalDate.now());  LocalDate tomorrow = LocalDate.now().withDayOfMonth(LocalDate.now().getDayOfMonth() + 1);  setEndDate(tomorrow);  setRefreshRate(100 + "");  min.textProperty().addListener(new ChangeListener<String>() {  @Override  public void changed(  ObservableValue<? extends String> observable,  String oldValue, String newValue) {  try {  int mi = Integer.parseInt(newValue);  int ma = Integer.parseInt(max.textProperty().get());  if (mi > ma) {  max.setText(mi + "");  }  min.setText(mi + "");  } catch (NumberFormatException e) {  min.setText(oldValue);  }  }  });  max.textProperty().addListener(new ChangeListener<String>() {  @Override  public void changed(  ObservableValue<? extends String> observable,  String oldValue, String newValue) {  try {  int ma = Integer.parseInt(newValue);  int mi = Integer.parseInt(min.textProperty().get());  if (ma < mi) {  min.setText(ma + "");  }  max.setText(ma + "");  } catch (NumberFormatException e) {  max.setText(oldValue);  }  }  });  startDate.valueProperty().addListener(  new ChangeListener<LocalDate>() {  @Override  public void changed(  ObservableValue<? extends LocalDate> observable,  LocalDate oldValue, LocalDate newValue) {  // TODO Auto-generated method stub  LocalDate end = endDate.valueProperty().get();  if (newValue.isBefore(LocalDate.now())) {  startDate.setValue(oldValue);  }  if (newValue.isAfter(end) || newValue.isEqual(end)) {  LocalDate tomorrow = newValue.withDayOfMonth(newValue.getDayOfMonth() + 1);  endDate.setValue(tomorrow);  }  }  });  endDate.valueProperty().addListener(  new ChangeListener<LocalDate>() {  @Override  public void changed(  ObservableValue<? extends LocalDate> observable,  LocalDate oldValue, LocalDate newValue) {  // TODO Auto-generated method stub  LocalDate start = startDate.valueProperty().get();  ///if (newValue.isBefore(LocalDate.now())) {  /// endDate.setValue(oldValue);  ///}  if (newValue.isBefore(start) || newValue.isEqual(start)) {  endDate.setValue(oldValue);  }  }  });  refreshRate.textProperty().addListener(  new ChangeListener<String>() {  @Override  public void changed(  ObservableValue<? extends String> observable,  String oldValue, String newValue) {  try {  int val = Integer.parseInt(newValue);  if (val > 100) {  refreshRate.setText(val + "");  } else {  refreshRate.setText(100 + "");  }  } catch (NumberFormatException e) {  refreshRate.setText(oldValue);  }  }  });  // title.setOnMouseClicked(new EventHandler<MouseEvent>() {  //  // @Override  // public void handle(MouseEvent event) {  // // TODO Auto-generated method stub  // MouseButton button = event.getButton();  // if (button == MouseButton.SECONDARY) {  // title.setTextFill(Color.BLACK);  // }  // }  // });  } catch (IOException exception) {  throw new RuntimeException(exception);  }  }  public void handlerAlertResolved(MouseEvent event) {  MouseButton button = event.getButton();  if (button == MouseButton.SECONDARY) {  title.setTextFill(Color.BLACK);  }  }  public void handlerKill(MouseEvent event) {  // if (selected) {  // boot.setText("Alive");  // boot.setTextFill(Color.GREEN);  // active.setDisable(false);  // active.setSelected(true);  // handlerActiveInactive(null);  //  // try {  // if (universal == null) {  // universal = new Socket("127.0.0.1", port);  // }  // if (encoder == null) {  // encoder = new MsgEncoder(universal.getOutputStream());  // }  //  // KeyValueList bootMsg = new KeyValueList();  // bootMsg.putPair("Scope", SCOPE);  // bootMsg.putPair("MessageType", "Boot");  // bootMsg.putPair("Name", getTitle());  //  // encoder.sendMsg(bootMsg);  //  // } catch (UnknownHostException e) {  // // TODO Auto-generated catch block  // e.printStackTrace();  // } catch (IOException e) {  // // TODO Auto-generated catch block  // e.printStackTrace();  // }  try {  if (universal == null) {  universal = new Socket("127.0.0.1", port);  }  if (encoder == null) {  encoder = new MsgEncoder(universal.getOutputStream());  }  KeyValueList kill = new KeyValueList();  kill.putPair("Scope", CreateGUI.SCOPE);  kill.putPair("MessageType", "Setting");  kill.putPair("Sender", CreateGUI.NAME);  kill.putPair("Receiver", getTitle());  kill.putPair("Purpose", "Kill");  encoder.sendMsg(kill);  active.setSelected(false);  handlerActiveInactive(null);  active.setDisable(true);  } catch (UnknownHostException e) {  // TODO Auto-generated catch block  e.printStackTrace();  } catch (IOException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  }  public void handlerActiveInactive(MouseEvent event) {  boolean selected = active.selectedProperty().get();  if (selected) {  active.setText("Active");  active.setTextFill(Color.GREEN);  try {  if (universal == null) {  universal = new Socket("127.0.0.1", port);  }  if (encoder == null) {  encoder = new MsgEncoder(universal.getOutputStream());  }  KeyValueList act = new KeyValueList();  act.putPair("Scope", CreateGUI.SCOPE);  act.putPair("MessageType", "Setting");  act.putPair("Sender", CreateGUI.NAME);  act.putPair("Receiver", getTitle());  act.putPair("Purpose", "Activate");  act.putPair("Max", getMax());  act.putPair("Min", getMin());  LocalDate st = getStartDate();  Instant instantS = st.atStartOfDay()  .atZone(ZoneId.systemDefault()).toInstant();  Date dateS = Date.from(instantS);  LocalDate ed = getEndDate();  Instant instantE = ed.atStartOfDay()  .atZone(ZoneId.systemDefault()).toInstant();  Date dateE = Date.from(instantE);  act.putPair("StartDate", dateS.getTime() + "");  act.putPair("EndDate", dateE.getTime() + "");  act.putPair("RefreshRate", getRefreshRate());  encoder.sendMsg(act);  } catch (UnknownHostException e) {  // TODO Auto-generated catch block  e.printStackTrace();  } catch (IOException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  } else {  active.setText("Inactive");  active.setTextFill(Color.RED);  try {  if (universal == null) {  universal = new Socket("127.0.0.1", port);  }  if (encoder == null) {  encoder = new MsgEncoder(universal.getOutputStream());  }  KeyValueList dact = new KeyValueList();  dact.putPair("Scope", CreateGUI.SCOPE);  dact.putPair("MessageType", "Setting");  dact.putPair("Sender", CreateGUI.NAME);  dact.putPair("Receiver", getTitle());  dact.putPair("Purpose", "Deactivate");  encoder.sendMsg(dact);  } catch (UnknownHostException e) {  // TODO Auto-generated catch block  e.printStackTrace();  } catch (IOException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  }  }  public void setAlert() {  title.setTextFill(Color.RED);  }  public void setTitle(String t) {  titleProperty().set(t);  }  public String getTitle() {  return titleProperty().get();  }  public StringProperty titleProperty() {  return title.textProperty();  }  public String getConsole() {  return consoleProperty().get();  }  public void setConsole(String value) {  consoleProperty().set(value);  }  public void setActive() {  active.setDisable(false);  active.setSelected(true);  active.setText("Active");  active.setTextFill(Color.GREEN);  }  public void setEnable() {  active.setDisable(false);  }  public StringProperty consoleProperty() {  return console.textProperty();  }  public String getMax() {  return maxProperty().get();  }  public void setMax(String value) {  maxProperty().set(value);  }  public StringProperty maxProperty() {  return max.textProperty();  }  public String getMin() {  return minProperty().get();  }  public void setMin(String value) {  minProperty().set(value);  }  public StringProperty minProperty() {  return min.textProperty();  }  public String getRefreshRate() {  return refreshRateProperty().get();  }  public void setRefreshRate(String value) {  refreshRateProperty().set(value);  }  public StringProperty refreshRateProperty() {  return refreshRate.textProperty();  }  public void setStartDate(LocalDate date) {  startDateProperty().set(date);  }  public LocalDate getStartDate() {  return startDateProperty().get();  }  public ObjectProperty<LocalDate> startDateProperty() {  return startDate.valueProperty();  }  public void setEndDate(LocalDate date) {  endDateProperty().set(date);  }  public LocalDate getEndDate() {  return endDateProperty().get();  }  public ObjectProperty<LocalDate> endDateProperty() {  return endDate.valueProperty();  }  public void setSorAItems(){  max.setDisable(true);  min.setDisable(true);  startDate.setDisable(true);  endDate.setDisable(true);  refreshRate.setDisable(true);  active.setSelected(true);  handlerActiveInactive(null);  active.setDisable(true);  }  public void setAudioItems(){  max.setDisable(true);  min.setDisable(true);  startDate.setDisable(false);  endDate.setDisable(false);  refreshRate.setDisable(false);  }  } |

CreateBreath.java

|  |
| --- |
| import java.io.BufferedReader;  import java.io.File;  import java.io.FileReader;  import java.io.IOException;  import java.io.InputStreamReader;  import java.net.Socket;  import java.text.ParseException;  import java.text.SimpleDateFormat;  import java.util.ArrayList;  import java.util.Arrays;  import java.util.Date;  import java.util.List;  import java.util.Timer;  import java.util.TimerTask;  import javax.swing.JFrame;  import javax.swing.SwingUtilities;  public class CreateBreath  {  // socket for connection to SISServer  static Socket universal;  private static int port = 53217;  // message writer  static MsgEncoder encoder;  // message reader  static MsgDecoder decoder;  // scope of this component  private static final String SCOPE = "SIS.Scope1";  // name of this component  private static final String NAME = "Breath";  // messages types that can be handled by this component  private static final List<String> TYPES = new ArrayList<String>(  Arrays.asList(new String[] { "Setting", "Confirm" }));  private static int refreshRate = 500;  private static Date startDate = new Date(), endDate = new Date();  private static Timer timer = new Timer();  //Breath custom variables.  private static boolean stopped = true;  // private static long prevtime = System.currentTimeMillis();  // shared by all kinds of records that can be generated by this component  private static KeyValueList record = new KeyValueList();  // shared by all kinds of alerts that can be generated by this component  private static KeyValueList alert = new KeyValueList();  private static BreathReading reading = new BreathReading();  private static final long RECORD\_TIME = 10000;  private static SimpleDateFormat format = new SimpleDateFormat(  "yyyy-MM-dd HH:mm:ss");  /\*  \* Main program  \*/  public static void main(String[] args)  {  while (true)  {  try  {  // try to establish a connection to SISServer  universal = connect();  // bind the message reader to inputstream of the socket  decoder = new MsgDecoder(universal.getInputStream());  // bind the message writer to outputstream of the socket  encoder = new MsgEncoder(universal.getOutputStream());  /\*  \* construct a Connect message to establish the connection  \*/  KeyValueList conn = new KeyValueList();  conn.putPair("Scope", SCOPE);  conn.putPair("MessageType", "Connect");  conn.putPair("Role", "Basic");  conn.putPair("Name", NAME);  encoder.sendMsg(conn);  initRecord();  // KeyValueList for inward messages, see KeyValueList for  // details  KeyValueList kvList;  while (true)  {  // attempt to read and decode a message, see MsgDecoder for  // details  kvList = decoder.getMsg();  // process that message  ProcessMsg(kvList);  }  }  catch (Exception e)  {  // if anything goes wrong, try to re-establish the connection  e.printStackTrace();  try  {  // wait for 1 second to retry  Thread.sleep(1000);  }  catch (InterruptedException e2)  {  }  System.out.println("Try to reconnect");  try  {  universal = connect();  }  catch (IOException e1)  {  }  }  }  }  /\*  \* used for connect(reconnect) to SISServer  \*/  static Socket connect() throws IOException  {  Socket socket = new Socket("127.0.0.1", port);  return socket;  }  private static void initRecord()  {  record.putPair("Scope", SCOPE);  record.putPair("MessageType", "Reading");  record.putPair("Sender", NAME);  // Receiver may be different for each message, so it doesn't make sense  // to set here  // record.putPair("Receiver", "");  alert.putPair("Scope", SCOPE);  alert.putPair("MessageType", "Alert");  alert.putPair("Sender", NAME);  alert.putPair("Purpose", "BreathAlert");  // Receiver may be different for each message, so it doesn't make sense  // to set here  // alert.putPair("Receiver", "");  }  private static void componentTask()  {  try  {  recordAudio();  if(System.currentTimeMillis() - reading.date < 5000) {  //reading  System.out.println("Reading: " + reading.decibel + " " + reading.date);  record.putPair("DecibelLevel", reading.decibel + "");  record.putPair("Date", reading.date + "");  //send reading message to GUI, uploader  encoder.sendMsg(record);  }else{  //alert  reading.date = System.currentTimeMillis();  System.out.println("Alert: " + reading.decibel + " " + reading.date);  alert.putPair("DecibelLevel", reading.decibel + "");  alert.putPair("Date", reading.date + "");  //send alert message to GUI, uploader  alert.removePair("Receiver");  alert.putPair("Receiver", "Uploader");  encoder.sendMsg(alert);  }  }  catch (Exception e)  {  e.printStackTrace();  }  }  private static void ProcessMsg(KeyValueList kvList) throws Exception  {  String scope = kvList.getValue("Scope");  if (!SCOPE.startsWith(scope))  {  return;  }  String messageType = kvList.getValue("MessageType");  if (!TYPES.contains(messageType))  {  return;  }  String sender = kvList.getValue("Sender");  String receiver = kvList.getValue("Receiver");  String purpose = kvList.getValue("Purpose");  switch (messageType)  {  case "Confirm":  System.out.println("Connect to SISServer successful.");  break;  case "Setting":  if (receiver.equals(NAME))  {  System.out.println("Message from " + sender);  System.out.println("Message type: " + messageType);  System.out.println("Message Purpose: " + purpose);  switch (purpose)  {  case "Activate":  String rRate = kvList.getValue("RefreshRate");  String sDate = kvList.getValue("StartDate");  String eDate = kvList.getValue("EndDate");  BreathDetector.main();  if (rRate != null && !rRate.equals(""))  {  refreshRate = Integer.parseInt(rRate);  }  if (sDate != null && !sDate.equals("") && eDate != null  && !eDate.equals(""))  {  startDate.setTime(Long.parseLong(sDate));  endDate.setTime(Long.parseLong(eDate));  }  try  {  timer.cancel();  timer = new Timer();  }  catch (Exception e)  {  // TODO: handle exception  }  timer.schedule(new TimerTask()  {  @Override  public void run()  {  // TODO Auto-generated method stub  if (System.currentTimeMillis() - endDate.getTime() > 0)  {  cancel();  }  else  {  componentTask();  }  }  }, startDate, refreshRate);  System.out.println("Algorithm Activated");  break;  case "Kill":  try  {  timer.cancel();  }  catch (Exception e)  {  // TODO: handle exception  }  System.exit(0);  break;  case "Deactivate":  try  {  timer.cancel();  BreathDetector.stopped = true;  BreathDetector.frame.setVisible(false);  BreathDetector.frame.dispose();  System.out.println("Recording stopped.");  }  catch (Exception e)  {  // TODO: handle exception  }  System.out.println("Algorithm Deactivated");  break;  }  }  break;  }  }  private static void recordAudio() throws Exception  {  try  {  reading.decibel = BreathDetector.breathLevel;  reading.date = BreathDetector.breathTime;  }  catch(Exception e) {}  // }  }  static class BreathReading  {  double decibel;  long date;  }  } |