Contents

1 AudioRender.java

```java
import java.io.File;
import java.util.HashMap;
import java.util.Map;

import javax.sound.midi.MidiSystem;
import javax.sound.midi.Receiver;
import javax.sound.midi.ShortMessage;
import javax.sound.sampled.AudioFileFormat;
import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;

import com.sun.media.sound.AudioSynthesizer;

public class AudioRender {

  public static void main(String[] args) throws Exception
  {
    /*
    * Open synthesizer in pull mode in the format 96000hz 24 bit stereo
    * using Sinc interpolation for highest quality.
    * With 1024 in max polyphony.
    */
    AudioFormat format = new AudioFormat(96000, 24, 2, true, false);
    AudioSynthesizer synthesizer = (AudioSynthesizer)MidiSystem.getSynthesizer();
    Map<String,Object> info = new HashMap<String,Object>();
    info.put("resampletType", "sinc");
    info.put("maxPolyphony", "1024");
    AudioInputStream stream = synthesizer.openStream(format, info);

    /*
    * Play midi note 60 on channel 1 for 1 sec.
    */
    ShortMessage msg = new ShortMessage();
    Receiver recv = synthesizer.getReceiver();
    msg.setMessage(ShortMessage.PROGRAM_CHANGE, 0, 48, 0);
    recv.send(msg, 0);
    msg.setMessage(ShortMessage.NOTE_ON, 0, 60, 80);
    recv.send(msg, 0);
    msg.setMessage(ShortMessage.NOTE_ON, 0, 60, 0);
    recv.send(msg, 1000000);

    /*
    * Calculate how many bytes 4 seconds are.
    */
    long len = (long)(format.getFrameRate() * 4);

    /*
    * Write 10 second into output file.
    */
    stream = new AudioInputStream(stream, format, len);
    AudioSystem.write(stream, AudioFileFormat.Type.WAVE, new File("output.wav"));

    /*
    * Close all resources.
    */
    synthesizer.close();
  }
}
```

2 FMTest1.java

```java
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

import javax.sound.midi.MidiSystem;
import javax.sound.midi.Sequence;
import javax.sound.midi.Sequencer;
import javax.sound.midi.Synthesizer;

import com.sun.media.sound.ModelAbstractOscillator;


public class FMTest1 {

  // A implementation of very simple FM Oscillator
  // ==============================================

  public static class MyOscillator extends ModelAbstractOscillator
  {
    double ix = 0;
    double last_ix_step = -1;

    public int read(float[][] buffers, int offset, int len) throws IOException {

      // Grab channel 0 buffer from buffers
      float[] buffer = buffers[0];

      // Calculate ix step so sin oscillirator is tuned so 6900 cents is 440 hz
      double target_ix_step =
        Math.exp((getPitch()-6900) * (Math.log(2.0) / 1200.0))
        * (440 / getSampleRate()) * (Math.PI*2);
      double ix_step = last_ix_step;
      if(ix_step == -1) ix_step = target_ix_step;
      double ix_step_step = (target_ix_step - ix_step)/len;

      // Simple FM synthesizer implementation
      int endoffset = offset + len;
      for (int i = offset; i < endoffset; i++) {
        buffer[i] = (float)Math.sin(ix + Math.sin(ix*3));
        ix += ix_step;
        // ix_step_step is used for
        // smooth pitch changes
        ix_step += ix_step_step;
      }

      last_ix_step = target_ix_step;

      return len;
    }


  }

  // This code is only for testing the instrument we defined/created above.
    // ========================================================================

  public static void main(String[] args) throws Exception {

    Synthesizer synth = MidiSystem.getSynthesizer();
    synth.open();
```

```
61      synth.unloadAllInstruments(synth.getDefaultSoundbank());
62      synth.loadAllInstruments(new MyOscillator());
63      Sequence seq = MidiSystem.getSequence(FMTest1.class.getResource("/FMTest1.mid"));
64      Sequencer seqr = MidiSystem.getSequencer(false);
65      seqr.open();
66      seqr.getTransmitter().setReceiver(synth.getReceiver());
67      seqr.setSequence(seq);
68      seqr.start();
69
70      System.out.println();
71      System.out.println("Is active, press enter to stop");
72      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
73      br.readLine();
74      System.out.println("Stop...");
75
76      seqr.stop();
77      seqr.close();
78      synth.close();
79
80      System.exit(0);
81    }
82
83  }
```

3 GeneratedInstrument1.java

```
1  import java.awt.BasicStroke;
2  import java.awt.BorderLayout;
3  import java.awt.Color;
4  import java.awt.Dimension;
5  import java.awt.FlowLayout;
6  import java.awt.Graphics;
7  import java.awt.Graphics2D;
8  import java.awt.GridBagConstraints;
9  import java.awt.GridBagLayout;
10 import java.awt.Insets;
11 import java.awt.Point;
12 import java.awt.event.ActionEvent;
13 import java.awt.event.ActionListener;
14 import java.awt.event.FocusEvent;
15 import java.awt.event.FocusListener;
16 import java.awt.event.KeyEvent;
17 import java.awt.event.KeyListener;
18 import java.awt.geom.Line2D;
19 import java.util.ArrayList;
20 import java.util.Arrays;
21 import java.util.HashMap;
22
23 import javax.sound.midi.MidiChannel;
24 import javax.sound.midi.MidiMessage;
25 import javax.sound.midi.Patch;
26 import javax.sound.midi.Receiver;
27 import javax.sound.midi.Soundbank;
28 import javax.sound.midi.Synthesizer;
29 import javax.sound.sampled.AudioFormat;
30 import javax.swing.BorderFactory;
31 import javax.swing.BoxLayout;
32 import javax.swing.JApplet;
33 import javax.swing.JButton;
34 import javax.swing.JCheckBox;
35 import javax.swing.JComboBox;
36 import javax.swing.JComponent;
37 import javax.swing.JLabel;
38 import javax.swing.JPanel;
39 import javax.swing.JScrollPane;
40 import javax.swing.JSlider;
41 import javax.swing.JTabbedPane;
42 import javax.swing.JTextField;
43 import javax.swing.SwingUtilities;
44 import javax.swing.event.ChangeEvent;
45 import javax.swing.event.ChangeListener;
46
47 import com.sun.media.sound.AudioFloatConverter;
48 import com.sun.media.sound.FFT;
49 import com.sun.media.sound.SF2Instrument;
50 import com.sun.media.sound.SF2InstrumentRegion;
51 import com.sun.media.sound.SF2Layer;
52 import com.sun.media.sound.SF2LayerRegion;
53 import com.sun.media.sound.SF2Region;
54 import com.sun.media.sound.SF2Sample;
55 import com.sun.media.sound.SF2Soundbank;
56 import com.sun.media.sound.SoftSynthesizer;
57
58 public class GeneratedInstrument1 extends JApplet {
59
60     float dbRange = 100;
```

```
61
62      ArrayList<SfGeneratorEditor> generators = new ArrayList<SfGeneratorEditor>();
63
64      private class Preset {
65          private String name;
66
67          private int[] gains = new int[gain_sliders.length];
68
69          private int[] widths = new int[gain_sliders.length];
70
71          private double[] harmonic = new double[gain_sliders.length];
72
73          private HashMap<Integer, Integer> gens = new HashMap<Integer, Integer>();
74
75          public Preset(String name) {
76              this.name = name;
77              gains[0] = 100;
78              for (int i = 0; i < harmonic.length; i++) {
79                  harmonic[i] = i + 1;
80              }
81          }
82
83          public String toString() {
84              return name;
85          }
86
87          public void setHarmonic(int id, int gain, int width) {
88              if (gain < -100)
89                  gain = -100;
90              if (gain > 0)
91                  gain = 0;
92              if (width < 0)
93                  width = 0;
94              if (width > 100)
95                  width = 100;
96              gains[id - 1] = 100 + gain;
97              widths[id - 1] = width;
98          }
99
100         public void setHarmonic(int id, double harmonic, int gain, int width) {
101             if (gain < -100)
102                 gain = -100;
103             if (gain > 0)
104                 gain = 0;
105             if (width < 0)
106                 width = 0;
107             if (width > 100)
108                 width = 100;
109             gains[id - 1] = 100 + gain;
110             widths[id - 1] = width;
111             this.harmonic[id - 1] = harmonic;
112         }
113
114         public void setGenerator(int genid, int value) {
115             gens.put(genid, value);
116         }
117
118         public void select() {
119             haltScreenUpdates = true;
120
121             for (int i = 0; i < gain_sliders.length; i++) {
122                 gain_sliders[i].setValue(gains[i]);
```

```
123             }
124             for (int i = 0; i < width_sliders.length; i++) {
125                 width_sliders[i].setValue(widths[i]);
126             }
127             for (int i = 0; i < width_sliders.length; i++) {
128                 if (Math.abs(harmonic[i] - (int) harmonic[i]) < 0.00001)
129                     harmonic_field[i].setText("" + (int) harmonic[i]);
130                 else
131                     harmonic_field[i].setText("" + harmonic[i]);
132             }
133 
134             for (SfGeneratorEditor gen : generators) {
135                 Integer v = gens.get(gen.gid);
136                 if (v != null)
137                     gen.setValue(v.intValue());
138                 else
139                     gen.setValue(gen.getDefaultValue());
140             }
141 
142             haltScreenUpdates = false;
143             dirty = true;
144             designInstrument(false);
145             sv.repaint();
146 
147         }
148 
149     }
150 
151     private class SfGeneratorEditor extends JPanel {
152 
153         private static final long serialVersionUID = 1L;
154 
155         private int gid;
156 
157         private JSlider slider;
158 
159         private JTextField textfield;
160 
161         private short getSFDefaultValue(int generator) {
162             if (generator == 8)
163                 return (short) 13500;
164             if (generator == 21)
165                 return (short) -12000;
166             if (generator == 23)
167                 return (short) -12000;
168             if (generator == 25)
169                 return (short) -12000;
170             if (generator == 26)
171                 return (short) -12000;
172             if (generator == 27)
173                 return (short) -12000;
174             if (generator == 28)
175                 return (short) -12000;
176             if (generator == 30)
177                 return (short) -12000;
178             if (generator == 33)
179                 return (short) -12000;
180             if (generator == 34)
181                 return (short) -12000;
182             if (generator == 35)
183                 return (short) -12000;
184             if (generator == 36)
```

```
            return (short) -12000;
        if (generator == 38)
            return (short) -12000;
        if (generator == 43)
            return (short) 0x7F00;
        if (generator == 44)
            return (short) 0x7F00;
        if (generator == 46)
            return (short) -1;
        if (generator == 47)
            return (short) -1;
        if (generator == 56)
            return (short) 100;
        if (generator == 58)
            return (short) -1;
        return 0;
    }

    public int getDefaultValue() {
        return getSFDefaultValue(gid);
    }

    private short getSFMinValue(int generator) {
        switch (generator) {
        case 0:
            return 0;
        case 1:
            return -32768;
        case 2:
            return -32768;
        case 3:
            return -32768;
        case 4:
            return 0;
        case 5:
            return -12000;
        case 6:
            return -12000;
        case 7:
            return -12000;
        case 8:
            return 1500;
        case 9:
            return 0;
        case 10:
            return -12000;
        case 11:
            return -12000;
        case 12:
            return -32768;
        case 13:
            return -960;
        case 15:
            return 0;
        case 16:
            return 0;
        case 17:
            return -500;
        case 21:
            return -16000;
        case 22:
            return -12000;
```

```
            case 23:
                return -16000;
            case 24:
                return -12000;
            case 25:
                return -12000;
            case 26:
                return -12000;
            case 27:
                return -12000;
            case 28:
                return -12000;
            case 29:
                return 0;
            case 30:
                return -12000;
            case 31:
                return -1200;
            case 32:
                return -1200;
            case 33:
                return -12000;
            case 34:
                return -12000;
            case 35:
                return -12000;
            case 36:
                return -12000;
            case 37:
                return 0;
            case 38:
                return -12000;
            case 39:
                return -1200;
            case 40:
                return -1200;
            case 43:
                return 0;
            case 44:
                return 0;
            case 45:
                return -32768;
            case 46:
                return 0;
            case 47:
                return 0;
            case 48:
                return 0;
            case 50:
                return -32768;
            case 51:
                return -120;
            case 52:
                return -99;
            case 54:
                return -32768;
            case 56:
                return 0;
            case 57:
                return 1;
            case 58:
                return 0;
```

```
309             default:
310                 return Short.MIN_VALUE;
311             }
312         }
313 
314         private short getSFMaxValue(int generator) {
315             switch (generator) {
316             case 0:
317                 return 32767;
318             case 1:
319                 return 0;
320             case 2:
321                 return 32767;
322             case 3:
323                 return 32767;
324             case 4:
325                 return 32767;
326             case 5:
327                 return 12000;
328             case 6:
329                 return 12000;
330             case 7:
331                 return 12000;
332             case 8:
333                 return 13500;
334             case 9:
335                 return 960;
336             case 10:
337                 return 12000;
338             case 11:
339                 return 12000;
340             case 12:
341                 return 0;
342             case 13:
343                 return 960;
344             case 15:
345                 return 1000;
346             case 16:
347                 return 1000;
348             case 17:
349                 return 500;
350             case 21:
351                 return 5000;
352             case 22:
353                 return 4500;
354             case 23:
355                 return 5000;
356             case 24:
357                 return 4500;
358             case 25:
359                 return 5000;
360             case 26:
361                 return 8000;
362             case 27:
363                 return 5000;
364             case 28:
365                 return 8000;
366             case 29:
367                 return 1000;
368             case 30:
369                 return 8000;
370             case 31:
```

```
            return 1200;
        case 32:
            return 1200;
        case 33:
            return 5000;
        case 34:
            return 8000;
        case 35:
            return 5000;
        case 36:
            return 8000;
        case 37:
            return 1440;
        case 38:
            return 8000;
        case 39:
            return 1200;
        case 40:
            return 1200;
        case 43:
            return 127;
        case 44:
            return 127;
        case 45:
            return 32767;
        case 46:
            return 127;
        case 47:
            return 127;
        case 48:
            return 1440;
        case 50:
            return 32767;
        case 51:
            return 120;
        case 52:
            return 99;
        case 54:
            return 32767;
        case 56:
            return 1200;
        case 57:
            return 127;
        case 58:
            return 127;
        default:
            return Short.MAX_VALUE;
        }
    }

    private String cent = "cent";

    private String cb = "0.1dB";

    private String timcent = "timcent";

    private String semitone = "semitone";

    private String mpercent = "0.1%";

    private String negative_mpercent = "-0.1%";

```

```
433         private String getSFType(int generator) {
434             switch (generator) {
435             case 5:
436                 return cent;
437             case 6:
438                 return cent;
439             case 7:
440                 return cent;
441             case 8:
442                 return cent;
443             case 9:
444                 return cb;
445             case 10:
446                 return cent;
447             case 11:
448                 return cent;
449             case 13:
450                 return cb;
451             case 15:
452                 return mpercent;
453             case 16:
454                 return mpercent;
455             case 17:
456                 return mpercent;
457             case 21:
458                 return timcent;
459             case 22:
460                 return cent;
461             case 23:
462                 return timcent;
463             case 24:
464                 return cent;
465             case 25:
466                 return timcent;
467             case 26:
468                 return timcent;
469             case 27:
470                 return timcent;
471             case 28:
472                 return timcent;
473             case 29:
474                 return negative_mpercent;
475             case 30:
476                 return timcent;
477             case 31:
478                 return timcent;
479             case 32:
480                 return timcent;
481             case 33:
482                 return timcent;
483             case 34:
484                 return timcent;
485             case 35:
486                 return timcent;
487             case 36:
488                 return timcent;
489             case 37:
490                 return cb;
491             case 38:
492                 return timcent;
493             case 39:
494                 return timcent;
```

```
            case 40:
                return timcent;
            case 48:
                return cb;
            case 51:
                return semitone;
            case 52:
                return cent;
            case 56:
                return cent;
            default:
                return "";
            }

        }

        public SfGeneratorEditor(final int gid) {
            this.gid = gid;

            setOpaque(false);
            setLayout(new FlowLayout(FlowLayout.LEFT, 2, 0));

            JButton defaulter = new JButton();
            defaulter.setText("R");
            defaulter.setMargin(new Insets(0, 0, 0, 0));
            defaulter.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent arg0) {
                    slider.setValue(getSFDefaultValue(gid));
                }
            });

            slider = new JSlider();
            slider.setOpaque(false);
            slider.setMinimum(getSFMinValue(gid));
            slider.setMaximum(getSFMaxValue(gid));
            slider.setValue(getSFDefaultValue(gid));
            slider.setMaximumSize(new Dimension(85, 20));
            slider.setPreferredSize(new Dimension(85, 20));
            slider.addChangeListener(new ChangeListener() {
                public void stateChanged(ChangeEvent arg0) {
                    dirty = true;
                    updateDisplay();
                }
            });
            add(slider);
            textfield = new JTextField("");
            textfield.setMaximumSize(new Dimension(50, 20));
            textfield.setPreferredSize(new Dimension(50, 20));
            textfield.addKeyListener(new KeyListener() {
                public void keyPressed(KeyEvent arg0) {
                    if (arg0.getKeyCode() == KeyEvent.VK_ENTER) {
                        slider.setValue(Integer.parseInt(textfield.getText()));
                        arg0.consume();
                    }
                }

                public void keyReleased(KeyEvent arg0) {
                }

                public void keyTyped(KeyEvent arg0) {
                }
            });
```

```
            textfield.addFocusListener(new FocusListener() {
                public void focusGained(FocusEvent arg0) {
                }

                public void focusLost(FocusEvent arg0) {
                    int x = Integer.parseInt(textfield.getText());
                    if (slider.getValue() != x)
                        slider.setValue(x);
                    updateDisplay();
                }
            });
            add(textfield);
            updateDisplay();
            add(defaulter);
            add(new JLabel(getSFType(gid)));
            generators.add(this);
        }

        public void setValue(int i) {
            if (slider.getValue() != i)
                slider.setValue(i);
        }

        public int getValue() {
            return slider.getValue();
        }

        private void updateDisplay() {
            if (textfield == null)
                return;
            textfield.setText("" + slider.getValue());
        }

        public void process(SF2Region region) {
            if (slider.getValue() != getSFDefaultValue(gid)) {
                region.putInteger(gid, slider.getValue());
            }
        }

    }

    private class SpecturmViewer extends JComponent {
        private static final long serialVersionUID = 1L;

        public void paint(Graphics g) {
            super.paint(g);

            int fftlen = data.length / 25;

            double base = 8 * 15;

            Graphics2D g2 = (Graphics2D) g;
            /*
             * g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
             * RenderingHints.VALUE_ANTIALIAS_ON);
             * g2.setRenderingHint(RenderingHints.KEY_FRACTIONALMETRICS,
             * RenderingHints.VALUE_FRACTIONALMETRICS_ON);
             */
            int w = getWidth();
            int h = getHeight() - 20;

            g2.setColor(Color.WHITE);
```

```
619             g2.fillRect(0, 0, w, h);
620
621             if (data != null) {
622
623                 g2.setStroke(new BasicStroke(1f));
624
625                 g2.setColor(Color.LIGHT_GRAY);
626                 for (int i = 0; i <= dbRange; i += 10) {
627                     float db = -i;
628                     float y = 10 + ((h - 20) * (-db / dbRange));
629                     if (y > (h - 10))
630                         y = h - 10;
631                     g2.draw(new Line2D.Float(20, y, w, y));
632                 }
633
634                 for (int i = 0; i <= gain_sliders.length; i++) {
635                     float x = (float) (20 + (i * base * (w - 20))
636                             / (float) fftlen);
637                     g2.draw(new Line2D.Float(x, 10, x, h - 10));
638                 }
639
640                 g2.setColor(Color.BLACK);
641                 for (int i = 10; i < dbRange; i += 10) {
642                     float db = -i;
643                     float y = 10 + ((h - 20) * (-db / dbRange));
644                     if (y > (h - 10))
645                         y = h - 10;
646                     g2.drawString("-" + i, 0, 4 + (int) y);
647                 }
648
649                 for (int i = 1; i <= gain_sliders.length; i++) {
650                     float x = (float) (20 + (i * base * (w - 20))
651                             / (float) fftlen);
652                     g2.drawString("" + i, x - 4, h + 3);
653                 }
654
655                 g2.setStroke(new BasicStroke(1.5f));
656
657                 float lastx = 20;
658                 float lasty = h - 10;
659                 for (int i = 0; i < fftlen; i++) {
660                     float x = 20 + (i * (w - 20)) / (float) fftlen;
661
662                     float db = (float) (20f * Math.log10(data[i * 2]));
663                     float y = 10 + ((h - 20) * (-db / dbRange));
664
665                     if (y > (h - 10))
666                         y = h - 10;
667                     g2.draw(new Line2D.Float(lastx, lasty, x, y));
668
669                     // g2.drawLine((int)lastx, (int)lasty, (int)x, y);
670                     lastx = x;
671                     lasty = y;
672                 }
673             }
674
675         }
676     }
677
678     private static final long serialVersionUID = 1L;
679
680     private Synthesizer synth = new SoftSynthesizer();
```

```
681
682     private Receiver recv;
683
684     boolean stereo_mode = false;
685
686     JPanel firstpanel;
687
688     JLabel infolabel;
689
690     String error_text = "";
691
692     JTextField[] harmonic_field = new JTextField[20];
693
694     JSlider[] width_sliders = new JSlider[20];
695
696     JSlider[] gain_sliders = new JSlider[20];
697
698     SpecturmViewer sv;
699
700     boolean dirty = false;
701
702     Soundbank sbk = null;
703
704     double[] data = null;
705
706     double[] data_audio;
707
708     double[] data_audio2;
709
710     ArrayList<Preset> presets = new ArrayList<Preset>();
711
712     boolean haltScreenUpdates = false;
713
714     public void complexGaussianDist(double[] cdata, double m, double s, double v) {
715         if (s < 0.5) {
716             int im = (int) m;
717             cdata[im * 2] = v;
718             return;
719         }
720         for (int x = 0; x < cdata.length / 4; x++) {
721             cdata[x * 2] += v
722                     * Math.exp((-1.0 / 2.0) * Math.pow((x - m) / s, 2.0));
723         }
724     }
725
726     public double[] realPart(double[] in) {
727         double[] out = new double[in.length / 2];
728         for (int i = 0; i < out.length; i++) {
729             out[i] = in[i * 2];
730         }
731         return out;
732     }
733
734     public void randomPhase(double[] data) {
735         for (int i = 0; i < data.length; i += 2) {
736             double phase = Math.random() * 2 * Math.PI;
737             double d = data[i];
738             data[i] = Math.sin(phase) * d;
739             data[i + 1] = Math.cos(phase) * d;
740         }
741     }
742
```

```
    FFT ifft_obj = null;

    int ifft_obj_len = 0;

    public void ifft(double[] data) {
        if (ifft_obj == null || ifft_obj_len != data.length / 2) {
            ifft_obj_len = data.length / 2;
            ifft_obj = new FFT(ifft_obj_len, 1);

        }
        ifft_obj.transform(data);
    }

    public SF2Sample newSimpleFFTSample(SF2Soundbank sf2, String name,
            double[] data, double base) {
        return newSimpleFFTSample(sf2, name, data, base, 10);
    }

    public float[] toFloat(double[] in) {
        float[] out = new float[in.length];
        for (int i = 0; i < out.length; i++) {
            out[i] = (float) in[i];
        }
        return out;
    }

    public float[] loopExtend(float[] data, int newsize) {
        float[] outdata = new float[newsize];
        int p_len = data.length;
        int p_ps = 0;
        for (int i = 0; i < outdata.length; i++) {
            outdata[i] = data[p_ps];
            p_ps++;
            if (p_ps == p_len)
                p_ps = 0;
        }
        return outdata;
    }

    public void normalize(double[] data, double target) {
        double maxvalue = 0;
        for (int i = 0; i < data.length; i++) {
            if (data[i] > maxvalue)
                maxvalue = data[i];
            if (-data[i] > maxvalue)
                maxvalue = -data[i];
        }
        if (maxvalue == 0)
            return;
        double gain = target / maxvalue;
        for (int i = 0; i < data.length; i++)
            data[i] *= gain;
    }

    public void fadeUp(float[] data, int samples) {
        double dsamples = samples;
        for (int i = 0; i < samples; i++)
            data[i] *= i / dsamples;
    }

    public byte[] toBytes(float[] in, AudioFormat format) {
        byte[] out = new byte[in.length * format.getFrameSize()];
```

```
        return AudioFloatConverter.getConverter(format).toByteArray(in, out);
    }

    public SF2Sample newSimpleFFTSample(SF2Soundbank sf2, String name,
            double[] data, double base, int fadeuptime) {

        int fftsize = data.length / 2;
        AudioFormat format = new AudioFormat(44100, 16, 1, true, false);
        double basefreq = (base / fftsize) * format.getSampleRate() * 0.5;

        randomPhase(data);
        ifft(data);
        data = realPart(data);
        normalize(data, 0.9);
        float[] fdata = toFloat(data);
        fdata = loopExtend(fdata, fdata.length + 512);
        fadeUp(fdata, fadeuptime);
        byte[] bdata = toBytes(fdata, format);

        /*
         * Create SoundFont2 sample.
         */
        SF2Sample sample = new SF2Sample(sf2);
        sample.setName(name);
        sample.setData(bdata);
        sample.setStartLoop(256);
        sample.setEndLoop(fftsize + 256);
        sample.setSampleRate((long) format.getSampleRate());
        double orgnote = (69 + 12)
                + (12 * Math.log(basefreq / 440.0) / Math.log(2));
        sample.setOriginalPitch((int) orgnote);
        sample.setPitchCorrection((byte) (-(orgnote - (int) orgnote) * 100.0));
        sf2.addResource(sample);

        return sample;
    }

    public Soundbank designInstrument(boolean createsbk) {

        int x = 8;
        int fftsize = 4096 * x;
        if (data == null || data.length != fftsize * 2)
            data = new double[fftsize * 2];

        Arrays.fill(data, 0);

        double[] data = this.data;
        double base = x * 15;

        for (int i = 0; i < width_sliders.length; i++) {
            double h = i + 1;
            try {
                h = Double.parseDouble(harmonic_field[i].getText());
            } catch (NumberFormatException e) {
                harmonic_field[i].setText("" + (i + 1));
            }
            if (gain_sliders[i].getValue() > 0) {
                double width = (width_sliders[i].getValue()) / 4;
                double db = -(100 - gain_sliders[i].getValue());
                double gain = Math.pow(10, db / 20.0);
                complexGaussianDist(data, base * h, width, gain);
            }
```

```
        }

        if (!createsbk)
            return null;

        SF2Soundbank sf2 = new SF2Soundbank();
        sf2.setName("My SoundFont");
        sf2.setVendor("Generated");
        sf2.setDescription("A newly created soundfont");

        SF2Layer layer1 = null;
        SF2Layer layer2 = null;

        for (int i = 0; i < 2; i++) {

            double[] data_audio = i == 0 ? this.data_audio : this.data_audio2;
            if (data_audio == null || data_audio.length != fftsize * 2) {
                data_audio = new double[fftsize * 2];
                if (i == 0)
                    this.data_audio = data_audio;
                else
                    this.data_audio2 = data_audio;
            }
            System.arraycopy(data, 0, data_audio, 0, data_audio.length);

            SF2Sample sample = newSimpleFFTSample(sf2, "FFT Sample " + i,
                    data_audio, base, 200);
            SF2Layer layer = newLayer(sf2, "FFT Layer " + i, sample);
            SF2Region region = layer.getRegions().get(0);
            region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1); // set loop
            if (stereo_mode)
                if (i == 0)
                    region.putInteger(SF2Region.GENERATOR_PAN, -500);
                else
                    region.putInteger(SF2Region.GENERATOR_PAN, 500);

            for (SfGeneratorEditor gen : generators) {
                gen.process(region);
            }

            if (i == 0)
                layer1 = layer;
            else
                layer2 = layer;

            if (!stereo_mode)
                break;

        }

        if (!stereo_mode)
            newInstrument(sf2, "FFT Instrument", new Patch(0, 0), layer1);
        else
            newInstrument(sf2, "FFT Instrument", new Patch(0, 0), layer1,
                    layer2);

        return sf2;

    }

    public SF2Layer newLayer(SF2Soundbank sf2, String name, SF2Sample sample) {
        SF2LayerRegion region = new SF2LayerRegion();
```

```
929         region.setSample(sample);
930 
931         SF2Layer layer = new SF2Layer(sf2);
932         layer.setName(name);
933         layer.getRegions().add(region);
934         sf2.addResource(layer);
935 
936         return layer;
937     }
938 
939     public SF2Instrument newInstrument(SF2Soundbank sf2, String name,
940             Patch patch, SF2Layer... layers) {
941 
942         /*
943          * Create SoundFont2 instrument.
944          */
945         SF2Instrument ins = new SF2Instrument(sf2);
946         ins.setPatch(patch);
947         ins.setName(name);
948         sf2.addInstrument(ins);
949 
950         /*
951          * Create region for instrument.
952          */
953         for (int i = 0; i < layers.length; i++) {
954             SF2InstrumentRegion insregion = new SF2InstrumentRegion();
955             insregion.setLayer(layers[i]);
956             ins.getRegions().add(insregion);
957         }
958 
959         return ins;
960     }
961 
962     public void destroy() {
963         synth.close();
964     }
965 
966     public void init() {
967         Runnable runnable = new Runnable() {
968             public void run() {
969                 try {
970 
971                     SwingUtilities.invokeAndWait(new Runnable() {
972                         public void run() {
973                             firstpanel = new JPanel();
974                             firstpanel.setBackground(Color.WHITE);
975                             firstpanel.setLayout(new FlowLayout());
976                             add(firstpanel);
977 
978                             infolabel = new JLabel(
979                                     "Loading␣synthesizer,␣please␣wait␣.␣.␣.");
980 
981                             firstpanel.add(infolabel);
982 
983                             validate();
984                             invalidate();
985                         }
986                     });
987 
988                     synth.getDefaultSoundbank();
989                     synth.open();
990                     recv = synth.getReceiver();
```

```
                SwingUtilities.invokeAndWait(new Runnable() {
                    public void run() {
                        createGUI();
                        validate();
                        invalidate();
                    }
                });

            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    };
    new Thread(runnable).start();
}

public void createGUI() {
    remove(firstpanel);
    JPanel toppanel = new JPanel();
    toppanel.setBackground(Color.WHITE);
    toppanel.setLayout(new FlowLayout());
    add(toppanel);

    JPanel boxpanel = new JPanel();
    boxpanel.setOpaque(false);
    boxpanel.setLayout(new BoxLayout(boxpanel, BoxLayout.Y_AXIS));
    toppanel.add(boxpanel);

    Dimension vdim12 = new Dimension(1000, 50);
    VirtualKeyboard12 vkeyboard12 = new VirtualKeyboard12();
    vkeyboard12.setSize(vdim12);
    vkeyboard12.setPreferredSize(vdim12);
    vkeyboard12.setMinimumSize(vdim12);
    vkeyboard12.setMaximumSize(vdim12);
    vkeyboard12.setChannel(0);

    final MidiChannel channel1 = synth.getChannels()[0];
    channel1.controlChange(7, 127);

    Receiver recv2 = new Receiver() {
        public void close() {
        }

        public void send(MidiMessage arg0, long arg1) {
            if (dirty) {
                synth.unloadAllInstruments(sbk);
                sbk = designInstrument(true);
                synth.loadAllInstruments(sbk);
                channel1.programChange(0);
                dirty = false;

            }
            recv.send(arg0, arg1);
        }
    };

    vkeyboard12.setReceiver(recv2);

    JScrollPane scrollpane12 = new JScrollPane(vkeyboard12);
    scrollpane12.setPreferredSize(new Dimension(500, 80));
    scrollpane12.getViewport().setViewPosition(new Point(200, 0));
```

```
        JPanel panel12 = new JPanel(new BorderLayout());
        panel12.setOpaque(false);
        panel12.add(scrollpane12);

        JPanel panelslides = new JPanel(new GridBagLayout());

        GridBagConstraints c = new GridBagConstraints();
        c.gridy = 0;
        c.gridx = 0;
        panelslides.add(new JLabel("Harmonic"), c);
        c.gridx = 1;
        panelslides.add(new JLabel("Amplitude"), c);
        c.gridx = 2;
        panelslides.add(new JLabel("Bandwidth"), c);

        final ChangeListener changelistener = new ChangeListener() {
            public void stateChanged(ChangeEvent arg0) {
                if (haltScreenUpdates)
                    return;
                dirty = true;
                if (sv != null) {
                    designInstrument(false);
                    sv.repaint();
                }
            }
        };

        for (int i = 0; i < width_sliders.length; i++) {

            JSlider amp_slider = new JSlider();
            amp_slider.setOpaque(false);
            amp_slider.setMinimum(0);
            amp_slider.setMaximum(100);
            amp_slider.setValue(0);
            amp_slider.addChangeListener(changelistener);

            JSlider w_slider = new JSlider();
            w_slider.setOpaque(false);
            w_slider.setMinimum(0);
            w_slider.setMaximum(100);
            w_slider.setValue(0);
            w_slider.addChangeListener(changelistener);

            JTextField tf = new JTextField("" + (i + 1));
            tf.setMaximumSize(new Dimension(50, 20));
            tf.setPreferredSize(new Dimension(50, 20));
            tf.addKeyListener(new KeyListener() {
                public void keyPressed(KeyEvent arg0) {
                    if (arg0.getKeyCode() == KeyEvent.VK_ENTER) {
                        changelistener.stateChanged(null);
                    }
                }

                public void keyReleased(KeyEvent arg0) {
                }

                public void keyTyped(KeyEvent arg0) {
                }
            });
            tf.addFocusListener(new FocusListener() {
                public void focusGained(FocusEvent arg0) {
```

```
            }

            public void focusLost(FocusEvent arg0) {
                changelistener.stateChanged(null);
            }
        });

        harmonic_field[i] = tf;
        width_sliders[i] = w_slider;
        gain_sliders[i] = amp_slider;

        c.gridy = i + 1;
        c.gridx = 0;
        // panelslides.add(new JLabel("" + (i + 1)), c);
        panelslides.add(tf, c);
        c.gridx = 1;
        panelslides.add(amp_slider, c);
        c.gridx = 2;
        panelslides.add(w_slider, c);

    }
    gain_sliders[0].setValue(100);
    GridBagConstraints ec = new GridBagConstraints();
    ec.gridy = 100;
    ec.gridwidth = 2;
    ec.weightx = 10;
    ec.weighty = 10;

    c.anchor = GridBagConstraints.LINE_START;
    JPanel volenv = new JPanel(new GridBagLayout());
    volenv.setBorder(BorderFactory.createTitledBorder("Volume Envelope"));
    volenv.setOpaque(false);
    c.gridy = 0;
    c.gridx = 0;
    volenv.add(new JLabel("Delay"), c);
    c.gridx = 1;
    volenv.add(new SfGeneratorEditor(SF2Region.GENERATOR_DELAYVOLENV), c);
    c.gridy = 1;
    c.gridx = 0;
    volenv.add(new JLabel("Attack"), c);
    c.gridx = 1;
    volenv.add(new SfGeneratorEditor(SF2Region.GENERATOR_ATTACKVOLENV), c);
    c.gridy = 2;
    c.gridx = 0;
    volenv.add(new JLabel("Hold"), c);
    c.gridx = 1;
    volenv.add(new SfGeneratorEditor(SF2Region.GENERATOR_HOLDVOLENV), c);
    c.gridy = 3;
    c.gridx = 0;
    volenv.add(new JLabel("Decay"), c);
    c.gridx = 1;
    volenv.add(new SfGeneratorEditor(SF2Region.GENERATOR_DECAYVOLENV), c);
    c.gridy = 4;
    c.gridx = 0;
    volenv.add(new JLabel("Sustain"), c);
    c.gridx = 1;
    volenv.add(new SfGeneratorEditor(SF2Region.GENERATOR_SUSTAINVOLENV), c);
    c.gridy = 5;
    c.gridx = 0;
    volenv.add(new JLabel("Release"), c);
    c.gridx = 1;
    volenv.add(new SfGeneratorEditor(SF2Region.GENERATOR_RELEASEVOLENV), c);
```

```
1177            c.gridy = 6;
1178            c.gridx = 0;
1179            volenv.add(new JLabel("Attenuation"), c);
1180            c.gridx = 1;
1181            volenv.add(
1182                    new SfGeneratorEditor(SF2Region.GENERATOR_INITIALATTENUATION),
1183                    c);
1184            volenv.add(new JLabel(), ec);
1185
1186            JPanel modenv = new JPanel(new GridBagLayout());
1187            modenv.setBorder(BorderFactory
1188                    .createTitledBorder("Modulation Envelope"));
1189            modenv.setOpaque(false);
1190            c.gridy = 0;
1191            c.gridx = 0;
1192            modenv.add(new JLabel("Delay"), c);
1193            c.gridx = 1;
1194            modenv.add(new SfGeneratorEditor(SF2Region.GENERATOR_DELAYMODENV), c);
1195            c.gridy = 1;
1196            c.gridx = 0;
1197            modenv.add(new JLabel("Attack"), c);
1198            c.gridx = 1;
1199            modenv.add(new SfGeneratorEditor(SF2Region.GENERATOR_ATTACKMODENV), c);
1200            c.gridy = 2;
1201            c.gridx = 0;
1202            modenv.add(new JLabel("Hold"), c);
1203            c.gridx = 1;
1204            modenv.add(new SfGeneratorEditor(SF2Region.GENERATOR_HOLDMODENV), c);
1205            c.gridy = 3;
1206            c.gridx = 0;
1207            modenv.add(new JLabel("Decay"), c);
1208            c.gridx = 1;
1209            modenv.add(new SfGeneratorEditor(SF2Region.GENERATOR_DECAYMODENV), c);
1210            c.gridy = 4;
1211            c.gridx = 0;
1212            modenv.add(new JLabel("Sustain"), c);
1213            c.gridx = 1;
1214            modenv.add(new SfGeneratorEditor(SF2Region.GENERATOR_SUSTAINMODENV), c);
1215            c.gridy = 5;
1216            c.gridx = 0;
1217            modenv.add(new JLabel("Release"), c);
1218            c.gridx = 1;
1219            modenv.add(new SfGeneratorEditor(SF2Region.GENERATOR_RELEASEMODENV), c);
1220            c.gridy = 6;
1221            c.gridx = 0;
1222            modenv.add(new JLabel("To Filter Cutoff"), c);
1223            c.gridx = 1;
1224            modenv.add(new SfGeneratorEditor(SF2Region.GENERATOR_MODENVTOFILTERFC),
1225                    c);
1226            c.gridy = 7;
1227            c.gridx = 0;
1228            modenv.add(new JLabel("To Pitch"), c);
1229            c.gridx = 1;
1230            modenv.add(new SfGeneratorEditor(SF2Region.GENERATOR_MODENVTOPITCH), c);
1231            modenv.add(new JLabel(), ec);
1232
1233            JPanel filter_gen = new JPanel(new GridBagLayout());
1234            filter_gen.setBorder(BorderFactory.createTitledBorder("Filter"));
1235            filter_gen.setOpaque(false);
1236            c.gridy = 0;
1237            c.gridx = 0;
1238            filter_gen.add(new JLabel("Cutoff Freq."), c);
```

```
1239            c.gridx = 1;
1240            filter_gen.add(new SfGeneratorEditor(
1241                    SF2Region.GENERATOR_INITIALFILTERFC), c);
1242            c.gridy = 1;
1243            c.gridx = 0;
1244            filter_gen.add(new JLabel("Q"), c);
1245            c.gridx = 1;
1246            filter_gen.add(
1247                    new SfGeneratorEditor(SF2Region.GENERATOR_INITIALFILTERQ), c);
1248            filter_gen.add(new JLabel(), ec);
1249
1250            JPanel lfo_mod = new JPanel(new GridBagLayout());
1251            lfo_mod.setBorder(BorderFactory.createTitledBorder("Modulation LFO"));
1252            lfo_mod.setOpaque(false);
1253            c.gridy = 0;
1254            c.gridx = 0;
1255            lfo_mod.add(new JLabel("Delay"), c);
1256            c.gridx = 1;
1257            lfo_mod.add(new SfGeneratorEditor(SF2Region.GENERATOR_DELAYMODLFO), c);
1258            c.gridy = 1;
1259            c.gridx = 0;
1260            lfo_mod.add(new JLabel("Freq"), c);
1261            c.gridx = 1;
1262            lfo_mod.add(new SfGeneratorEditor(SF2Region.GENERATOR_FREQMODLFO), c);
1263            c.gridy = 2;
1264            c.gridx = 0;
1265            lfo_mod.add(new JLabel("To Filter Cutoff"), c);
1266            c.gridx = 1;
1267            lfo_mod.add(
1268                    new SfGeneratorEditor(SF2Region.GENERATOR_MODLFOTOFILTERFC), c);
1269            c.gridy = 3;
1270            c.gridx = 0;
1271            lfo_mod.add(new JLabel("To Pitch"), c);
1272            c.gridx = 1;
1273            lfo_mod
1274                    .add(new SfGeneratorEditor(SF2Region.GENERATOR_MODLFOTOPITCH),
1275                            c);
1276            lfo_mod.add(new JLabel(), ec);
1277
1278            JPanel lfo_vib = new JPanel(new GridBagLayout());
1279            lfo_vib.setBorder(BorderFactory.createTitledBorder("Vibration LFO"));
1280            lfo_vib.setOpaque(false);
1281            c.gridy = 0;
1282            c.gridx = 0;
1283            lfo_vib.add(new JLabel("Delay"), c);
1284            c.gridx = 1;
1285            lfo_vib.add(new SfGeneratorEditor(SF2Region.GENERATOR_DELAYVIBLFO), c);
1286            c.gridy = 1;
1287            c.gridx = 0;
1288            lfo_vib.add(new JLabel("Freq"), c);
1289            c.gridx = 1;
1290            lfo_vib.add(new SfGeneratorEditor(SF2Region.GENERATOR_FREQVIBLFO), c);
1291            c.gridy = 2;
1292            c.gridx = 0;
1293            lfo_vib.add(new JLabel("To Pitch"), c);
1294            c.gridx = 1;
1295            lfo_vib
1296                    .add(new SfGeneratorEditor(SF2Region.GENERATOR_VIBLFOTOPITCH),
1297                            c);
1298            lfo_vib.add(new JLabel(), ec);
1299
1300            GridBagConstraints c2 = new GridBagConstraints();
```

```
        c2.fill = GridBagConstraints.BOTH;
        JPanel generators = new JPanel(new GridBagLayout());
        generators.setBorder(BorderFactory.createEmptyBorder(2, 2, 2, 2));
        generators.setOpaque(false);
        c2.gridx = 0;
        c2.gridy = 0;
        generators.add(volenv, c2);
        c2.gridx = 1;
        c2.gridy = 0;
        generators.add(modenv, c2);
        c2.gridx = 0;
        c2.gridy = 1;
        generators.add(lfo_vib, c2);
        c2.gridx = 1;
        c2.gridy = 1;
        generators.add(lfo_mod, c2);
        c2.gridx = 0;
        c2.gridy = 2;
        generators.add(filter_gen, c2);
        generators.add(new JLabel(), ec);

        sbk = designInstrument(true);

        synth.unloadAllInstruments(synth.getDefaultSoundbank());
        synth.loadAllInstruments(sbk);
        channel1.programChange(0);

        panelslides.setOpaque(false);
        panelslides.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));

        sv = new SpecturmViewer();
        Dimension dim = new Dimension(300, 200);
        sv.setMinimumSize(dim);
        sv.setPreferredSize(dim);

        initPresets();
        final JComboBox presetSelector = new JComboBox(presets.toArray());
        presetSelector.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent arg0) {
                ((Preset) presetSelector.getSelectedItem()).select();
            }

        });

        JPanel instrumentpanel = new JPanel();
        instrumentpanel.setLayout(new FlowLayout(FlowLayout.LEFT));
        instrumentpanel.setOpaque(false);
        instrumentpanel.add(new JLabel("Preset: "));
        instrumentpanel.add(presetSelector);
        final JCheckBox reberbcheckbox = new JCheckBox("Reverb");
        reberbcheckbox.setOpaque(false);
        reberbcheckbox.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                channel1.controlChange(91, reberbcheckbox.isSelected() ? 127
                        : 0);
            }
        });
        channel1.controlChange(91, 0);
        instrumentpanel.add(reberbcheckbox);
        final JCheckBox choruscheckbox = new JCheckBox("Chorus");
        choruscheckbox.setOpaque(false);
```

```
        choruscheckbox.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                channel1.controlChange(93, choruscheckbox.isSelected() ? 127
                        : 0);
            }
        });
        channel1.controlChange(93, 0);
        instrumentpanel.add(choruscheckbox);
        final JCheckBox portamentocheckbox = new JCheckBox("Portamento");
        portamentocheckbox.setOpaque(false);
        portamentocheckbox.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {

                if (portamentocheckbox.isSelected()) {
                    channel1.controlChange(126, 1); // Mono Mode
                    channel1.controlChange(65, 127); // Set Portamento On
                    channel1.controlChange(5, 67); // Set portamento time
                } else {
                    channel1.controlChange(127, 1); // Poly Mode
                    channel1.controlChange(65, 0); // Set Portamento off
                }
            }
        });
        instrumentpanel.add(portamentocheckbox);

        final JCheckBox stereo_mode_checkbox = new JCheckBox("Stereo");
        stereo_mode_checkbox.setOpaque(false);
        stereo_mode_checkbox.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                changelistener.stateChanged(null);
                stereo_mode = stereo_mode_checkbox.isSelected();
            }
        });
        instrumentpanel.add(stereo_mode_checkbox);

        boxpanel.add(instrumentpanel);
        boxpanel.add(panel12);
        boxpanel.add(sv);

        JPanel presets_panel = new JPanel();
        presets_panel.setLayout(new BoxLayout(presets_panel, BoxLayout.Y_AXIS));
        presets_panel.setOpaque(false);

        JTabbedPane tabs = new JTabbedPane();
        tabs.addTab("Harmonic Profile", panelslides);
        tabs.addTab("Instrument Settings", generators);
        boxpanel.add(tabs);

    }

    public void initPresets() {
        Preset preset;

        preset = new Preset("Sine wave");
        presets.add(preset);

        preset = new Preset("Square wave");
        for (int i = 1; i <= 20; i += 2) {
            double x = 1.0 / (double) i;
            preset.setHarmonic(i, (int) (20 * Math.log10(x)), 0);
        }
        presets.add(preset);
```

```
1425
1426        preset = new Preset("Triangle wave");
1427        for (int i = 1; i <= 20; i += 2) {
1428            double x = 1.0 / (double) (i * i);
1429            preset.setHarmonic(i, (int) (20 * Math.log10(x)), 0);
1430        }
1431        presets.add(preset);
1432        preset = new Preset("Sawtooth wave");
1433        for (int i = 1; i <= 20; i++) {
1434            double x = 1.0 / (double) i;
1435            preset.setHarmonic(i, (int) (20 * Math.log10(x)), 0);
1436        }
1437        presets.add(preset);
1438
1439        preset = new Preset("Synth Piano");
1440        preset.setHarmonic(1, 0, 2);
1441        preset.setHarmonic(2, -20, 3);
1442        preset.setHarmonic(3, -40, 4);
1443        preset.setHarmonic(4, -15, 5);
1444        preset.setHarmonic(5, -10, 6);
1445        preset.setHarmonic(6, -40, 7);
1446        preset.setHarmonic(7, -35, 8);
1447        preset.setHarmonic(8, -45, 9);
1448        preset.setHarmonic(9, -55, 10);
1449        preset.setHarmonic(10, -60, 11);
1450        preset.setHarmonic(11, -65, 12);
1451        preset.setHarmonic(12, -55, 13);
1452        preset.setHarmonic(13, -65, 14);
1453        preset.setHarmonic(14, -70, 15);
1454        preset.setGenerator(SF2Region.GENERATOR_ATTACKVOLENV, -12000);
1455        preset.setGenerator(SF2Region.GENERATOR_RELEASEVOLENV, -1000);
1456        preset.setGenerator(SF2Region.GENERATOR_DECAYVOLENV, 4000);
1457        preset.setGenerator(SF2Region.GENERATOR_SUSTAINVOLENV, 1440);
1458        preset.setGenerator(SF2Region.GENERATOR_INITIALFILTERFC, 9500);
1459        presets.add(preset);
1460
1461        preset = new Preset("Flute");
1462        preset.setHarmonic(1, 0, 0);
1463        preset.setHarmonic(2, 0, 0);
1464        for (int i = 2; i < 20; i++) {
1465            preset.setHarmonic(i + 1, -(i-2) * 5 - 10, 0);
1466        }
1467        preset.setGenerator(SF2Region.GENERATOR_ATTACKVOLENV, -6000);
1468        preset.setGenerator(SF2Region.GENERATOR_RELEASEVOLENV, -1000);
1469        preset.setGenerator(SF2Region.GENERATOR_DECAYVOLENV, 4000);
1470        preset.setGenerator(SF2Region.GENERATOR_SUSTAINVOLENV, -100);
1471        preset.setGenerator(SF2Region.GENERATOR_INITIALFILTERFC, 9500);
1472        presets.add(preset);
1473
1474
1475        preset = new Preset("Flute 2");
1476        preset.setHarmonic(1, 0, 0);
1477        preset.setHarmonic(2, -25, 0);
1478        preset.setHarmonic(3, -15, 0);
1479        preset.setHarmonic(4, -30, 0);
1480        preset.setHarmonic(5, -25, 0);
1481        preset.setHarmonic(6, -40, 0);
1482        preset.setHarmonic(7, -35, 0);
1483        preset.setHarmonic(8, -45, 0);
1484        preset.setHarmonic(9, -55, 0);
1485        preset.setHarmonic(10, -60, 0);
1486        preset.setHarmonic(11, -65, 0);
```

```
        preset.setHarmonic(12, -55, 0);
        preset.setHarmonic(13, -65, 0);
        preset.setHarmonic(14, -70, 0);
        preset.setGenerator(SF2Region.GENERATOR_ATTACKVOLENV, -6000);
        preset.setGenerator(SF2Region.GENERATOR_RELEASEVOLENV, -1000);
        preset.setGenerator(SF2Region.GENERATOR_DECAYVOLENV, 4000);
        preset.setGenerator(SF2Region.GENERATOR_SUSTAINVOLENV, -100);
        preset.setGenerator(SF2Region.GENERATOR_INITIALFILTERFC, 9500);
        presets.add(preset);

        preset = new Preset("Trumpet");
        preset.setHarmonic(1, -20, 0);
        preset.setHarmonic(2, -15, 0);
        preset.setHarmonic(3, -8, 0);
        for (int i = 3; i < 20; i++) {
            preset.setHarmonic(i + 1, -i * 4, 0);
        }
        preset.setGenerator(SF2Region.GENERATOR_ATTACKVOLENV, -10000);
        preset.setGenerator(SF2Region.GENERATOR_RELEASEVOLENV, 0);
        preset.setGenerator(SF2Region.GENERATOR_DECAYVOLENV, 4000);
        preset.setGenerator(SF2Region.GENERATOR_SUSTAINVOLENV, -100);

        preset.setGenerator(SF2Region.GENERATOR_ATTACKMODENV, -4000);
        preset.setGenerator(SF2Region.GENERATOR_RELEASEMODENV, -2500);
        preset.setGenerator(SF2Region.GENERATOR_MODENVTOFILTERFC, 5000);
        preset.setGenerator(SF2Region.GENERATOR_INITIALFILTERFC, 4500);
        preset.setGenerator(SF2Region.GENERATOR_INITIALFILTERQ, 10);

        presets.add(preset);

        preset = new Preset("Horn");
        preset.setHarmonic(1, -10, 0);
        for (int i = 1; i < 20; i++) {
            preset.setHarmonic(i + 1, -i * 3, 0);
        }
        preset.setGenerator(SF2Region.GENERATOR_SAMPLEMODES, 1);
        preset.setGenerator(SF2Region.GENERATOR_ATTACKVOLENV, -6000);
        preset.setGenerator(SF2Region.GENERATOR_RELEASEVOLENV, -1000);
        preset.setGenerator(SF2Region.GENERATOR_DECAYVOLENV, 4000);
        preset.setGenerator(SF2Region.GENERATOR_SUSTAINVOLENV, -100);

        preset.setGenerator(SF2Region.GENERATOR_ATTACKMODENV, -500);
        preset.setGenerator(SF2Region.GENERATOR_RELEASEMODENV, 12000);
        preset.setGenerator(SF2Region.GENERATOR_MODENVTOFILTERFC, 5000);
        preset.setGenerator(SF2Region.GENERATOR_INITIALFILTERFC, 4500);

        presets.add(preset);

        preset = new Preset("Strings");
        for (int i = 0; i < 20; i++) {
            preset.setHarmonic(i + 1, -i * 4, i * 4 + 5);
        }
        preset.setGenerator(SF2Region.GENERATOR_ATTACKVOLENV, -4000);
        preset.setGenerator(SF2Region.GENERATOR_RELEASEVOLENV, 2000);

        presets.add(preset);

        preset = new Preset("Choir");
        for (int i = 0; i < 20; i++) {
            preset.setHarmonic(i + 1, -i * 4, i * 4 + 10);
        }
        preset.setHarmonic(5 + 1, -40, 5 * 4);
```

```
1549            preset.setHarmonic(6 + 1, -50, 6 * 4);
1550            preset.setHarmonic(7 + 1, -60, 7 * 4);
1551            preset.setHarmonic(8 + 1, -70, 8 * 4);
1552
1553            preset.setGenerator(SF2Region.GENERATOR_ATTACKVOLENV, -4000);
1554            preset.setGenerator(SF2Region.GENERATOR_RELEASEVOLENV, 2000);
1555
1556            presets.add(preset);
1557
1558            preset = new Preset("Choir␣2");
1559            for (int i = 0; i < 4; i++) {
1560                preset.setHarmonic(i + 1, -i * 7, i * 4 + 10);
1561            }
1562            for (int i = 10; i < 20; i++) {
1563                preset.setHarmonic(i + 1, -i * 4, i * 4 + 10);
1564            }
1565            preset.setGenerator(SF2Region.GENERATOR_ATTACKVOLENV, -4000);
1566            preset.setGenerator(SF2Region.GENERATOR_RELEASEVOLENV, 2000);
1567            presets.add(preset);
1568
1569            preset = new Preset("Bell");
1570            for (int i = 0; i < 20; i++) {
1571                preset.setHarmonic(i + 1, (i + 1) + (i == 0 ? 0 : 0.05 + (i == 1 ? 0.15 : 0)), -i *
                        5,
1572                        i * 1 + 5);
1573            }
1574            preset.setGenerator(SF2Region.GENERATOR_SUSTAINVOLENV, 1440);
1575            preset.setGenerator(SF2Region.GENERATOR_RELEASEVOLENV, 4000);
1576            preset.setGenerator(SF2Region.GENERATOR_DECAYVOLENV, 4000);
1577            preset.setGenerator(SF2Region.GENERATOR_INITIALFILTERFC, 9500);
1578
1579            presets.add(preset);
1580
1581            preset = new Preset("Musical␣Bell");
1582            for (int i = 0; i < 20; i++) {
1583                preset.setHarmonic(i + 1, (i + 1) + (i == 0 ? 0 : 0.2), (i == 0 ? 0
1584                        : -35)
1585                        + (i == 16 ? 30 : 0)
1586                        + (int) (20 * Math.log10(1.0 / (i * i + 1.0))),
1587                        (i == 16 ? 20 : 0) + i);
1588            }
1589            preset.setGenerator(SF2Region.GENERATOR_SUSTAINVOLENV, 1440);
1590            preset.setGenerator(SF2Region.GENERATOR_RELEASEVOLENV, 4000);
1591            preset.setGenerator(SF2Region.GENERATOR_DECAYVOLENV, 4000);
1592
1593            presets.add(preset);
1594
1595            preset = new Preset("New␣Age");
1596            for (int i = 0; i < 20; i++) {
1597                preset.setHarmonic(i + 1, -i * 4, i * 4 + 5);
1598            }
1599            preset.setHarmonic(20, -30, 85);
1600            preset.setGenerator(SF2Region.GENERATOR_ATTACKVOLENV, -4000);
1601            preset.setGenerator(SF2Region.GENERATOR_RELEASEVOLENV, 4000);
1602            preset.setGenerator(SF2Region.GENERATOR_DECAYVOLENV, -1200);
1603            preset.setGenerator(SF2Region.GENERATOR_SUSTAINVOLENV, 150);
1604
1605            presets.add(preset);
1606
1607        }
1608
1609    }
```

4 MakeSoundFont.java

```java
/*
 * Copyright (c) 2007 by Karl Helgason
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * - Redistributions of source code must retain the above copyright notice,
 *   this list of conditions and the following disclaimer.
 * - Redistributions in binary form must reproduce the above copyright
 *   notice, this list of conditions and the following disclaimer in the
 *   documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 */

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

import javax.sound.sampled.*;

import com.sun.media.sound.*;

public class MakeSoundFont {

  public static void main(String[] args)
      throws UnsupportedAudioFileException, IOException {
    /*
     * Create new SoundFont2 soundbank
     */
    SF2Soundbank sf2 = new SF2Soundbank();

    /*
     * Select audio file.
     */
    File audiofile = new File("ding.wav");
    AudioInputStream audiosream = AudioSystem
        .getAudioInputStream(audiofile);

    /*
     * Make sure the audio stream is in
     * correct format for soundfonts
     * e.g.16 bit signed, little endian
     */
    AudioFormat format = new AudioFormat(audiosream.getFormat()
        .getSampleRate(), 16, 1, true, false);
    AudioInputStream convaudiosream = AudioSystem.getAudioInputStream(
        format, audiosream);
```

```
61
62      /*
63       * Read the content of the file into a byte array.
64       */
65      int datalength = (int) convaudiosream.getFrameLength()
66          * format.getFrameSize();
67      byte[] data = new byte[datalength];
68      convaudiosream.read(data, 0, data.length);
69      audiosream.close();
70
71      /*
72       * Create SoundFont2 sample.
73       */
74      SF2Sample sample = new SF2Sample(sf2);
75      sample.setName("Ding␣Sample");
76      sample.setData(data);
77      sample.setSampleRate((long) format.getSampleRate());
78      sample.setOriginalPitch(75);
79      sf2.addResource(sample);
80
81      /*
82       * Create SoundFont2 layer.
83       */
84      SF2Layer layer = new SF2Layer(sf2);
85      layer.setName("Ding␣Layer");
86      sf2.addResource(layer);
87
88      /*
89       * Create region for layer.
90       */
91      SF2LayerRegion region = new SF2LayerRegion();
92      region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 12000);
93      region.setSample(sample);
94      layer.getRegions().add(region);
95
96      /*
97       * Create SoundFont2 instrument.
98       */
99      SF2Instrument ins = new SF2Instrument(sf2);
100     ins.setName("Ding␣Instrument");
101     sf2.addInstrument(ins);
102
103     /*
104      * Create region for instrument.
105      */
106     SF2InstrumentRegion insregion = new SF2InstrumentRegion();
107     insregion.setLayer(layer);
108     ins.getRegions().add(insregion);
109
110     /*
111      * Save soundbank to disk.
112      */
113     sf2.save("ding.sf2");
114   }
115
116 }
```

5 Midi2WavRender.java

```java
/*
 * Copyright (c) 2007 by Karl Helgason
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * - Redistributions of source code must retain the above copyright notice,
 *   this list of conditions and the following disclaimer.
 * - Redistributions in binary form must reproduce the above copyright
 *   notice, this list of conditions and the following disclaimer in the
 *   documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 */

import java.io.File;
import java.io.FileNotFoundException;

import javax.sound.midi.MetaMessage;
import javax.sound.midi.MidiDevice;
import javax.sound.midi.MidiEvent;
import javax.sound.midi.MidiMessage;
import javax.sound.midi.MidiSystem;
import javax.sound.midi.MidiUnavailableException;
import javax.sound.midi.Receiver;
import javax.sound.midi.Sequence;
import javax.sound.midi.Soundbank;
import javax.sound.midi.Synthesizer;
import javax.sound.midi.Track;
import javax.sound.midi.MidiDevice.Info;
import javax.sound.sampled.AudioFileFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;

import com.sun.media.sound.AudioSynthesizer;

public class Midi2WavRender {

  public static void main(String[] args) {
    if (args.length >= 2)
      try {
        File midi_file = new File(args[0]);
        if (!midi_file.exists())
          throw new FileNotFoundException();
        Sequence sequence = MidiSystem.getSequence(midi_file);
        Soundbank soundbank = null;
        if (args.length >= 3) {
```

```
61          File soundbank_file = new File(args[2]);
62          if (soundbank_file.exists())
63            soundbank = MidiSystem.getSoundbank(soundbank_file);
64        }
65        render(soundbank, sequence, new File(args[1]));
66        System.exit(0);
67
68      } catch (Exception e) {
69        System.out.println(e.toString());
70        System.out.println();
71      }
72    System.out.println("MIDI to WAVE Render: usages:");
73    System.out
74        .println("java Midi2WavRender <midi_file_in> <wave_file_out> <soundbank_file>");
75    System.exit(1);
76  }
77
78  /*
79   * Render sequence using selected or default soundbank into wave audio file.
80   */
81  public static void render(Soundbank soundbank, Sequence sequence,
82      File audio_file) {
83    try {
84      // Find available AudioSynthesizer.
85      AudioSynthesizer synth = findAudioSynthesizer();
86      if (synth == null) {
87        System.out.println("No AudioSynhtesizer was found!");
88        System.exit(1);
89      }
90
91      // Open AudioStream from AudioSynthesizer.
92      AudioInputStream stream = synth.openStream(null, null);
93
94      // Load user-selected Soundbank into AudioSynthesizer.
95      if (soundbank != null) {
96        Soundbank defsbk = synth.getDefaultSoundbank();
97        if (defsbk != null)
98          synth.unloadAllInstruments(defsbk);
99        synth.loadAllInstruments(soundbank);
100     }
101
102     // Play Sequence into AudioSynthesizer Receiver.
103     double total = send(sequence, synth.getReceiver());
104
105     // Calculate how long the WAVE file needs to be.
106     long len = (long) (stream.getFormat().getFrameRate() * (total + 4));
107     stream = new AudioInputStream(stream, stream.getFormat(), len);
108
109     // Write WAVE file to disk.
110     AudioSystem.write(stream, AudioFileFormat.Type.WAVE, audio_file);
111
112     // We are finished, close synthesizer.
113     synth.close();
114   } catch (Exception e) {
115     e.printStackTrace();
116   }
117 }
118
119 /*
120  * Find available AudioSynthesizer.
121  */
122 public static AudioSynthesizer findAudioSynthesizer()
```

```
123       throws MidiUnavailableException {
124     // First check if default synthesizer is AudioSynthesizer.
125     Synthesizer synth = MidiSystem.getSynthesizer();
126     if (synth instanceof AudioSynthesizer)
127       return (AudioSynthesizer) synth;
128 
129     // If default synhtesizer is not AudioSynthesizer, check others.
130     Info[] infos = MidiSystem.getMidiDeviceInfo();
131     for (int i = 0; i < infos.length; i++) {
132       MidiDevice dev = MidiSystem.getMidiDevice(infos[i]);
133       if (dev instanceof AudioSynthesizer)
134         return (AudioSynthesizer) dev;
135     }
136 
137     // No AudioSynthesizer was found, return null.
138     return null;
139   }
140 
141   /*
142    * Send entiry MIDI Sequence into Receiver using timestamps.
143    */
144   public static double send(Sequence seq, Receiver recv) {
145     float divtype = seq.getDivisionType();
146     assert (seq.getDivisionType() == Sequence.PPQ);
147     Track[] tracks = seq.getTracks();
148     int[] trackspos = new int[tracks.length];
149     int mpq = 500000;
150     int seqres = seq.getResolution();
151     long lasttick = 0;
152     long curtime = 0;
153     while (true) {
154       MidiEvent selevent = null;
155       int seltrack = -1;
156       for (int i = 0; i < tracks.length; i++) {
157         int trackpos = trackspos[i];
158         Track track = tracks[i];
159         if (trackpos < track.size()) {
160           MidiEvent event = track.get(trackpos);
161           if (selevent == null
162               || event.getTick() < selevent.getTick()) {
163             selevent = event;
164             seltrack = i;
165           }
166         }
167       }
168       if (seltrack == -1)
169         break;
170       trackspos[seltrack]++;
171       long tick = selevent.getTick();
172       if (divtype == Sequence.PPQ)
173         curtime += ((tick - lasttick) * mpq) / seqres;
174       else
175         curtime = (long) ((tick * 1000000.0 * divtype) / seqres);
176       lasttick = tick;
177       MidiMessage msg = selevent.getMessage();
178       if (msg instanceof MetaMessage) {
179         if (divtype == Sequence.PPQ)
180           if (((MetaMessage) msg).getType() == 0x51) {
181             byte[] data = ((MetaMessage) msg).getData();
182             mpq = ((data[0] & 0xff) << 16)
183                 | ((data[1] & 0xff) << 8) | (data[2] & 0xff);
184           }
```

```
185         } else {
186           if (recv != null)
187             recv.send(msg, curtime);
188         }
189       }
190       return curtime / 1000000.0;
191     }
192
193 }
```

6 ReverbAndChorusApplet.java

```
1  import java.awt.BorderLayout;
2  import java.awt.Color;
3  import java.awt.Dimension;
4  import java.awt.FlowLayout;
5  import java.awt.GridBagConstraints;
6  import java.awt.GridBagLayout;
7  import java.awt.Insets;
8  import java.awt.Point;
9  import java.awt.event.ActionEvent;
10 import java.awt.event.ActionListener;
11 import java.io.IOException;
12
13 import javax.sound.midi.InvalidMidiDataException;
14 import javax.sound.midi.MidiMessage;
15 import javax.sound.midi.Patch;
16 import javax.sound.midi.Receiver;
17 import javax.sound.midi.ShortMessage;
18 import javax.sound.midi.Soundbank;
19 import javax.sound.midi.Synthesizer;
20 import javax.swing.BorderFactory;
21 import javax.swing.BoxLayout;
22 import javax.swing.JApplet;
23 import javax.swing.JComboBox;
24 import javax.swing.JLabel;
25 import javax.swing.JPanel;
26 import javax.swing.JScrollPane;
27 import javax.swing.JSlider;
28 import javax.swing.SwingUtilities;
29 import javax.swing.event.ChangeEvent;
30 import javax.swing.event.ChangeListener;
31
32 import com.sun.media.sound.SoftSynthesizer;
33
34 public class ReverbAndChorusApplet extends JApplet {
35
36     private static final long serialVersionUID = 1L;
37
38     private Synthesizer synth = new SoftSynthesizer();
39
40     private Receiver recv;
41
42     JPanel firstpanel;
43     JLabel infolabel;
44     String error_text = "";
45
46     public void destroy() {
47         synth.close();
48     }
49
50     public void init() {
51         Runnable runnable = new Runnable() {
52             public void run() {
53                 try {
54
55                     SwingUtilities.invokeAndWait(new Runnable() {
56                         public void run() {
57                             firstpanel = new JPanel();
58                             firstpanel.setBackground(Color.WHITE);
59                             firstpanel.setLayout(new FlowLayout());
60                             add(firstpanel);
```

```
                        infolabel = new JLabel(
                                "Loading synthesizer, please wait...");

                        firstpanel.add(infolabel);

                        validate();
                        invalidate();
                    }
                });

                synth.getDefaultSoundbank();
                synth.open();
                recv = synth.getReceiver();

                SwingUtilities.invokeAndWait(new Runnable() {
                    public void run() {
                        createGUI();
                        validate();
                        invalidate();
                    }
                });

            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    };
    new Thread(runnable).start();
}

public JSlider createMidiControlSlider(final int control, int defaultvalue)
{
    final JSlider slider = new JSlider(JSlider.HORIZONTAL);
    slider.setMaximum(127);
    slider.setMinimum(0);
    slider.setValue(defaultvalue);
    slider.addChangeListener(new ChangeListener()
    {
        public void stateChanged(ChangeEvent e) {
            try {
                ShortMessage sms = new ShortMessage();
                sms.setMessage(ShortMessage.CONTROL_CHANGE, control, slider.getValue());
                recv.send(sms, -1);
            } catch (InvalidMidiDataException e1) {
                e1.printStackTrace();
            }

        }
    });
    slider.setOpaque(false);
    return slider;
}

public JComboBox createReverbTypeCombobox()
{
    String[] reverbtypes = {"Small Room", "Medium Room", "Large Room", "Medium Hall", "Large
        Hall", "Plate"};
    final JComboBox combobox = new JComboBox(reverbtypes);
    combobox.setSelectedIndex(4);
    combobox.addActionListener(new ActionListener()
    {
```

```
122             public void actionPerformed(ActionEvent e) {
123                 int reverbType = 0;
124                 if(combobox.getSelectedIndex() == 0) reverbType = UniversalSysExBuilder.
                        DeviceControl.ReverbEffect.REVERB_TYPE_SMALL_ROOM;
125                 if(combobox.getSelectedIndex() == 1) reverbType = UniversalSysExBuilder.
                        DeviceControl.ReverbEffect.REVERB_TYPE_MEDIUM_ROOM;
126                 if(combobox.getSelectedIndex() == 2) reverbType = UniversalSysExBuilder.
                        DeviceControl.ReverbEffect.REVERB_TYPE_LARGE_ROOM;
127                 if(combobox.getSelectedIndex() == 3) reverbType = UniversalSysExBuilder.
                        DeviceControl.ReverbEffect.REVERB_TYPE_MEDIUM_HALL;
128                 if(combobox.getSelectedIndex() == 4) reverbType = UniversalSysExBuilder.
                        DeviceControl.ReverbEffect.REVERB_TYPE_LARGE_HALL;
129                 if(combobox.getSelectedIndex() == 5) reverbType = UniversalSysExBuilder.
                        DeviceControl.ReverbEffect.REVERB_TYPE_PLATE;
130                 try {
131                     MidiMessage msg = UniversalSysExBuilder.DeviceControl.ReverbEffect.
                            setReverbType(
132                                 UniversalSysExBuilder.ALL_DEVICES, reverbType);
133                     recv.send(msg, -1);
134                 } catch (IOException e1) {
135                     e1.printStackTrace();
136                 } catch (InvalidMidiDataException e1) {
137                     e1.printStackTrace();
138                 }
139             }
140         });
141         return combobox;
142     }
143 
144     public JSlider createReverbTimeSlider()
145     {
146         final JSlider slider = new JSlider(JSlider.HORIZONTAL);
147         slider.setMaximum(127);
148         slider.setMinimum(0);
149         slider.setValue(64);
150         slider.addChangeListener(new ChangeListener()
151         {
152             public void stateChanged(ChangeEvent e) {
153                 try {
154                     MidiMessage msg = UniversalSysExBuilder.DeviceControl.ReverbEffect.
                            setReverbTime(
155                                 UniversalSysExBuilder.ALL_DEVICES, slider.getValue());
156                     recv.send(msg, -1);
157                 } catch (IOException e1) {
158                     e1.printStackTrace();
159                 } catch (InvalidMidiDataException e1) {
160                     e1.printStackTrace();
161                 }
162 
163             }
164         });
165         slider.setOpaque(false);
166         return slider;
167     }
168 
169     public JComboBox createChorusTypeCombobox()
170     {
171         String[] chorustypes = {"Chorus 1", "Chorus 2", "Chorus 3", "Chorus 4", "FB Chorus", "
                Flanger"};
172         final JComboBox combobox = new JComboBox(chorustypes);
173         combobox.setSelectedIndex(2);
174         combobox.addActionListener(new ActionListener()
```

```
175         {
176             public void actionPerformed(ActionEvent e) {
177                 int chorusType = 0;
178                 if(combobox.getSelectedIndex() == 0) chorusType = UniversalSysExBuilder.
                        DeviceControl.ChorusEffect.CHORUS_TYPE_CHORUS1;
179                 if(combobox.getSelectedIndex() == 1) chorusType = UniversalSysExBuilder.
                        DeviceControl.ChorusEffect.CHORUS_TYPE_CHORUS2;
180                 if(combobox.getSelectedIndex() == 2) chorusType = UniversalSysExBuilder.
                        DeviceControl.ChorusEffect.CHORUS_TYPE_CHORUS3;
181                 if(combobox.getSelectedIndex() == 3) chorusType = UniversalSysExBuilder.
                        DeviceControl.ChorusEffect.CHORUS_TYPE_CHORUS4;
182                 if(combobox.getSelectedIndex() == 4) chorusType = UniversalSysExBuilder.
                        DeviceControl.ChorusEffect.CHORUS_TYPE_FB_CHORUS;
183                 if(combobox.getSelectedIndex() == 5) chorusType = UniversalSysExBuilder.
                        DeviceControl.ChorusEffect.CHORUS_TYPE_FLANGER;
184                 try {
185                     MidiMessage msg = UniversalSysExBuilder.DeviceControl.ChorusEffect.
                            setChorusType(
186                                 UniversalSysExBuilder.ALL_DEVICES, chorusType);
187                     recv.send(msg, -1);
188                 } catch (IOException e1) {
189                     e1.printStackTrace();
190                 } catch (InvalidMidiDataException e1) {
191                     e1.printStackTrace();
192                 }
193             }
194         });
195         return combobox;
196     }
197 
198     public JSlider createChorusFeedbackSlider()
199     {
200         final JSlider slider = new JSlider(JSlider.HORIZONTAL);
201         slider.setMaximum(127);
202         slider.setMinimum(0);
203         slider.setValue(64);
204         slider.addChangeListener(new ChangeListener()
205         {
206             public void stateChanged(ChangeEvent e) {
207                 try {
208                     MidiMessage msg = UniversalSysExBuilder.DeviceControl.ChorusEffect.
                            setChorusFeedback(
209                                 UniversalSysExBuilder.ALL_DEVICES, slider.getValue());
210                     recv.send(msg, -1);
211                 } catch (IOException e1) {
212                     e1.printStackTrace();
213                 } catch (InvalidMidiDataException e1) {
214                     e1.printStackTrace();
215                 }
216 
217             }
218         });
219         slider.setOpaque(false);
220         return slider;
221     }
222 
223     public JSlider createChorusModDepthSlider()
224     {
225         final JSlider slider = new JSlider(JSlider.HORIZONTAL);
226         slider.setMaximum(127);
227         slider.setMinimum(0);
228         slider.setValue(64);
```

```
229         slider.addChangeListener(new ChangeListener()
230         {
231             public void stateChanged(ChangeEvent e) {
232                 try {
233                     MidiMessage msg = UniversalSysExBuilder.DeviceControl.ChorusEffect.
                        setChorusModDepth(
234                             UniversalSysExBuilder.ALL_DEVICES, slider.getValue());
235                     recv.send(msg, -1);
236                 } catch (IOException e1) {
237                     e1.printStackTrace();
238                 } catch (InvalidMidiDataException e1) {
239                     e1.printStackTrace();
240                 }
241 
242             }
243         });
244         slider.setOpaque(false);
245         return slider;
246     }
247 
248     public JSlider createChorusModRateSlider()
249     {
250         final JSlider slider = new JSlider(JSlider.HORIZONTAL);
251         slider.setMaximum(127);
252         slider.setMinimum(0);
253         slider.setValue(64);
254         slider.addChangeListener(new ChangeListener()
255         {
256             public void stateChanged(ChangeEvent e) {
257                 try {
258                     MidiMessage msg = UniversalSysExBuilder.DeviceControl.ChorusEffect.
                        setChorusModRate(
259                             UniversalSysExBuilder.ALL_DEVICES, slider.getValue());
260                     recv.send(msg, -1);
261                 } catch (IOException e1) {
262                     e1.printStackTrace();
263                 } catch (InvalidMidiDataException e1) {
264                     e1.printStackTrace();
265                 }
266 
267             }
268         });
269         slider.setOpaque(false);
270         return slider;
271     }
272 
273     public JSlider createChorusSendToReverbSlider()
274     {
275         final JSlider slider = new JSlider(JSlider.HORIZONTAL);
276         slider.setMaximum(127);
277         slider.setMinimum(0);
278         slider.setValue(64);
279         slider.addChangeListener(new ChangeListener()
280         {
281             public void stateChanged(ChangeEvent e) {
282                 try {
283                     MidiMessage msg = UniversalSysExBuilder.DeviceControl.ChorusEffect.
                        setChorusSendToReverb(
284                             UniversalSysExBuilder.ALL_DEVICES, slider.getValue());
285                     recv.send(msg, -1);
286                 } catch (IOException e1) {
287                     e1.printStackTrace();
```

```
                } catch (InvalidMidiDataException e1) {
                    e1.printStackTrace();
                }

            }
        });
        slider.setOpaque(false);
        return slider;
    }

    public void createGUI() {
        remove(firstpanel);
        JPanel toppanel = new JPanel();
        toppanel.setBackground(Color.WHITE);
        toppanel.setLayout(new FlowLayout());
        add(toppanel);

        JPanel boxpanel = new JPanel();
        boxpanel.setOpaque(false);
        boxpanel.setLayout(new BoxLayout(boxpanel, BoxLayout.Y_AXIS));
        toppanel.add(boxpanel);

        Soundbank sbk = synth.getDefaultSoundbank();
        String[] instruments = new String[128];
        for (int i = 0; i < instruments.length; i++) {
            instruments[i] = i + "␣" + sbk.getInstrument(new Patch(0, i)).getName();
        }
        final JComboBox instrumentcombobox = new JComboBox(instruments);
        instrumentcombobox.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e) {
                int pgm = instrumentcombobox.getSelectedIndex();
                synth.getChannels()[0].programChange(pgm);
            }
        });


        JPanel instrumentpanel = new JPanel();
        instrumentpanel.setLayout(new FlowLayout(FlowLayout.LEFT));
        instrumentpanel.setOpaque(false);
        instrumentpanel.add(new JLabel("Instrument:␣"));
        instrumentpanel.add(instrumentcombobox);
        boxpanel.add(instrumentpanel);

        JPanel effectscontainer = new JPanel();
        effectscontainer.setLayout(new BoxLayout(effectscontainer, BoxLayout.X_AXIS));
        effectscontainer.setOpaque(false);
        boxpanel.add(effectscontainer);

        GridBagConstraints c = new GridBagConstraints();
        c.anchor = GridBagConstraints.WEST;
        c.insets = new Insets(2,2,2,2);

        JPanel reverbeffect = new JPanel();
        reverbeffect.setBorder(BorderFactory
                .createTitledBorder("Reverb"));
        reverbeffect.setOpaque(false);
        reverbeffect.setLayout(new GridBagLayout());

        c.gridx = 0;
        c.gridy = 0;
        reverbeffect.add(new JLabel("Level"),c);
```

```
        c.gridx = 1;
        reverbeffect.add(createMidiControlSlider(91, 40),c);

        c.gridx = 0;
        c.gridy = 1;
        reverbeffect.add(new JLabel("Type"),c);
        c.gridx = 1;
        c.weightx = 1;
        reverbeffect.add(createReverbTypeCombobox(),c);

        c.gridx = 0;
        c.gridy = 2;
        reverbeffect.add(new JLabel("Room size"),c);
        c.gridx = 1;
        reverbeffect.add(createReverbTimeSlider(),c);

        c.gridx = 0;
        c.gridy = 3;
        c.weighty = 1;
        JPanel emptypanel = new JPanel();
        emptypanel.setOpaque(false);
        reverbeffect.add(emptypanel, c);
        c.weighty = 0;

        effectscontainer.add(reverbeffect);

        JPanel choruseffect = new JPanel();
        choruseffect.setBorder(BorderFactory
                .createTitledBorder("Chorus"));
        choruseffect.setOpaque(false);
        choruseffect.setLayout(new GridBagLayout());

        c.gridx = 0;
        c.gridy = 0;
        choruseffect.add(new JLabel("Level"),c);
        c.gridx = 1;
        choruseffect.add(createMidiControlSlider(93, 0),c);

        c.gridx = 0;
        c.gridy = 1;
        choruseffect.add(new JLabel("Type"),c);
        c.gridx = 1;
        choruseffect.add(createChorusTypeCombobox(),c);

        c.gridx = 0;
        c.gridy = 2;
        choruseffect.add(new JLabel("Mod Rate"),c);
        c.gridx = 1;
        choruseffect.add(createChorusModRateSlider(),c);

        c.gridx = 0;
        c.gridy = 3;
        choruseffect.add(new JLabel("Mod Depth"),c);
        c.gridx = 1;
        choruseffect.add(createChorusModDepthSlider(),c);

        c.gridx = 0;
        c.gridy = 4;
        choruseffect.add(new JLabel("Feedback"),c);
        c.gridx = 1;
        choruseffect.add(createChorusFeedbackSlider(),c);

```

```
412         c.gridx = 0;
413         c.gridy = 5;
414         choruseffect.add(new JLabel("Send to Reverb"),c);
415         c.gridx = 1;
416         choruseffect.add(createChorusSendToReverbSlider(),c);
417 
418         effectscontainer.add(choruseffect);
419 
420 
421         Dimension vdim12 = new Dimension(1000, 50);
422         VirtualKeyboard12 vkeyboard12 = new VirtualKeyboard12();
423         vkeyboard12.setSize(vdim12);
424         vkeyboard12.setPreferredSize(vdim12);
425         vkeyboard12.setMinimumSize(vdim12);
426         vkeyboard12.setMaximumSize(vdim12);
427         vkeyboard12.setChannel(0);
428         vkeyboard12.setReceiver(recv);
429 
430         JScrollPane scrollpane12 = new JScrollPane(vkeyboard12);
431         scrollpane12.setPreferredSize(new Dimension(500, 80));
432         scrollpane12.getViewport().setViewPosition(new Point(200, 0));
433 
434         JPanel panel12 = new JPanel(new BorderLayout());
435         panel12.setOpaque(false);
436         panel12.add(scrollpane12);
437         boxpanel.add(panel12);
438 
439 
440     }
441 
442 }
```

7 SimpleApplet1.java

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Point;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.sound.midi.Patch;
import javax.sound.midi.Receiver;
import javax.sound.midi.Soundbank;
import javax.sound.midi.Synthesizer;
import javax.swing.BoxLayout;
import javax.swing.JApplet;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.SwingUtilities;

import com.sun.media.sound.SoftSynthesizer;

public class SimpleApplet1 extends JApplet {

    private static final long serialVersionUID = 1L;

    private Synthesizer synth = new SoftSynthesizer();

    private Receiver recv;

    JPanel firstpanel;
    JLabel infolabel;
    String error_text = "";

    public void destroy() {
        synth.close();
    }

    public void init() {
        Runnable runnable = new Runnable() {
            public void run() {
                try {

                    SwingUtilities.invokeAndWait(new Runnable() {
                        public void run() {
                            firstpanel = new JPanel();
                            firstpanel.setBackground(Color.WHITE);
                            firstpanel.setLayout(new FlowLayout());
                            add(firstpanel);

                            infolabel = new JLabel(
                                    "Loading synthesizer, please wait...");

                            firstpanel.add(infolabel);

                            validate();
                            invalidate();
                        }
                    });

```

```
                synth.getDefaultSoundbank();
                synth.open();
                recv = synth.getReceiver();

                SwingUtilities.invokeAndWait(new Runnable() {
                    public void run() {
                        createGUI();
                        validate();
                        invalidate();
                    }
                });

            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    };
    new Thread(runnable).start();
}

public void createGUI() {
    remove(firstpanel);
    JPanel toppanel = new JPanel();
    toppanel.setBackground(Color.WHITE);
    toppanel.setLayout(new FlowLayout());
    add(toppanel);

    JPanel boxpanel = new JPanel();
    boxpanel.setOpaque(false);
    boxpanel.setLayout(new BoxLayout(boxpanel, BoxLayout.Y_AXIS));
    toppanel.add(boxpanel);

    Soundbank sbk = synth.getDefaultSoundbank();
    String[] instruments = new String[128];
    for (int i = 0; i < instruments.length; i++) {
        instruments[i] = i + "␣" + sbk.getInstrument(new Patch(0, i)).getName();
    }
    final JComboBox instrumentcombobox = new JComboBox(instruments);
    instrumentcombobox.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent e) {
            int pgm = instrumentcombobox.getSelectedIndex();
            synth.getChannels()[0].programChange(pgm);
        }
    });
    JPanel instrumentpanel = new JPanel();
    instrumentpanel.setLayout(new FlowLayout(FlowLayout.LEFT));
    instrumentpanel.setOpaque(false);
    instrumentpanel.add(new JLabel("Instrument:␣"));
    instrumentpanel.add(instrumentcombobox);
    boxpanel.add(instrumentpanel);

    Dimension vdim12 = new Dimension(1000, 50);
    VirtualKeyboard12 vkeyboard12 = new VirtualKeyboard12();
    vkeyboard12.setSize(vdim12);
    vkeyboard12.setPreferredSize(vdim12);
    vkeyboard12.setMinimumSize(vdim12);
    vkeyboard12.setMaximumSize(vdim12);
    vkeyboard12.setChannel(0);
    vkeyboard12.setReceiver(recv);

    JScrollPane scrollpane12 = new JScrollPane(vkeyboard12);
```

```
123         scrollpane12.setPreferredSize(new Dimension(500, 80));
124         scrollpane12.getViewport().setViewPosition(new Point(200, 0));
125 
126         JPanel panel12 = new JPanel(new BorderLayout());
127         panel12.setOpaque(false);
128         panel12.add(scrollpane12);
129         boxpanel.add(panel12);
130 
131 
132     }
133 
134 }
```

8 TuningApplet1.java

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Point;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;

import javax.sound.midi.InvalidMidiDataException;
import javax.sound.midi.Patch;
import javax.sound.midi.Receiver;
import javax.sound.midi.ShortMessage;
import javax.sound.midi.Soundbank;
import javax.sound.midi.Synthesizer;
import javax.sound.midi.SysexMessage;
import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
import javax.swing.JApplet;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.SwingUtilities;

import com.sun.media.sound.SoftSynthesizer;

public class TuningApplet1 extends JApplet {

    private static final long serialVersionUID = 1L;

    private Synthesizer synth = new SoftSynthesizer();

    private Receiver recv;

    JPanel firstpanel;
    JLabel infolabel;
    String error_text = "";

    public void destroy() {
        synth.close();
    }

    public void init() {
        Runnable runnable = new Runnable() {
            public void run() {
                try {

                    SwingUtilities.invokeAndWait(new Runnable() {
                        public void run() {
                            firstpanel = new JPanel();
                            firstpanel.setBackground(Color.WHITE);
                            firstpanel.setLayout(new FlowLayout());
                            add(firstpanel);

                            infolabel = new JLabel(
                                    "Loading␣synthesizer,␣please␣wait␣.␣.␣.");

                            firstpanel.add(infolabel);

```

```
                        validate();
                        invalidate();
                    }
                });

                synth.getDefaultSoundbank();
                synth.open();
                recv = synth.getReceiver();

                SwingUtilities.invokeAndWait(new Runnable() {
                    public void run() {
                        createGUI();
                        validate();
                        invalidate();
                    }
                });

            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    };
    new Thread(runnable).start();
}

public static void sendTuningChange(Receiver recv, int channel,
        int tuningpreset) throws InvalidMidiDataException {
    // Data Entry
    ShortMessage sm1 = new ShortMessage();
    sm1.setMessage(ShortMessage.CONTROL_CHANGE, channel, 0x64, 03);
    ShortMessage sm2 = new ShortMessage();
    sm2.setMessage(ShortMessage.CONTROL_CHANGE, channel, 0x65, 00);
    // Tuning program 19
    ShortMessage sm3 = new ShortMessage();
    sm3
            .setMessage(ShortMessage.CONTROL_CHANGE, channel, 0x06,
                    tuningpreset);

    // Data Increment
    ShortMessage sm4 = new ShortMessage();
    sm4.setMessage(ShortMessage.CONTROL_CHANGE, channel, 0x60, 0x7F);
    // Data Decrement
    ShortMessage sm5 = new ShortMessage();
    sm5.setMessage(ShortMessage.CONTROL_CHANGE, channel, 0x61, 0x7F);

    recv.send(sm1, -1);
    recv.send(sm2, -1);
    recv.send(sm3, -1);
    recv.send(sm4, -1);
    recv.send(sm5, -1);
}

public static void sendTunings(Receiver recv, int bank, int preset,
        String name, double[] tunings) throws IOException,
        InvalidMidiDataException {
    int[] itunings = new int[128];
    for (int i = 0; i < itunings.length; i++) {
        itunings[i] = (int) (tunings[i] * 16384.0 / 100.0);
    }
    SysexMessage msg = UniversalSysExBuilder.MidiTuningStandard
            .keyBasedTuningDump(UniversalSysExBuilder.ALL_DEVICES, bank,
                    preset, name, itunings);
```

```
123         recv.send(msg, -1);
124     }
125 
126     public void createGUI() {
127         remove(firstpanel);
128         JPanel toppanel = new JPanel();
129         toppanel.setBackground(Color.WHITE);
130         toppanel.setLayout(new FlowLayout());
131         add(toppanel);
132 
133         JPanel boxpanel = new JPanel();
134         boxpanel.setOpaque(false);
135         boxpanel.setLayout(new BoxLayout(boxpanel, BoxLayout.Y_AXIS));
136         toppanel.add(boxpanel);
137 
138         Soundbank sbk = synth.getDefaultSoundbank();
139         String[] instruments = new String[128];
140         for (int i = 0; i < instruments.length; i++) {
141             instruments[i] = i + "␣" + sbk.getInstrument(new Patch(0, i)).getName();
142         }
143         final JComboBox instrumentcombobox = new JComboBox(instruments);
144         instrumentcombobox.addActionListener(new ActionListener()
145         {
146             public void actionPerformed(ActionEvent e) {
147                 int pgm = instrumentcombobox.getSelectedIndex();
148                 synth.getChannels()[0].programChange(pgm);
149                 synth.getChannels()[1].programChange(pgm);
150                 synth.getChannels()[2].programChange(pgm);
151                 synth.getChannels()[3].programChange(pgm);
152             }
153         });
154         JPanel instrumentpanel = new JPanel();
155         instrumentpanel.setLayout(new FlowLayout(FlowLayout.LEFT));
156         instrumentpanel.setOpaque(false);
157         instrumentpanel.add(new JLabel("Instrument:␣"));
158         instrumentpanel.add(instrumentcombobox);
159         boxpanel.add(instrumentpanel);
160 
161         try {
162 
163             double[] tunings = new double[128];
164 
165             for (int i = 0; i < tunings.length; i++)
166                 tunings[i] = 2400 + i * 1200.0 / 19.0;
167             sendTunings(recv, 0, 19, "19-TET", tunings);
168             sendTuningChange(recv, 0, 19);
169 
170             for (int i = 0; i < tunings.length; i++)
171                 tunings[i] = -2400 + i * 1200.0 / 7.0;
172             sendTunings(recv, 0, 7, "7-TET", tunings);
173             sendTuningChange(recv, 2, 7);
174 
175             for (int i = 0; i < tunings.length; i++)
176                 tunings[i] = -2400 + i * 1200.0 / 5.0;
177             sendTunings(recv, 0, 5, "5-TET", tunings);
178             sendTuningChange(recv, 3, 5);
179 
180         } catch (Exception e1) {
181             e1.printStackTrace();
182             return;
183         }
184 
```

```
        Dimension vdim19 = new Dimension(627, 50);
        VirtualKeyboard19 vkeyboard19 = new VirtualKeyboard19();
        vkeyboard19.setSize(vdim19);
        vkeyboard19.setPreferredSize(vdim19);
        vkeyboard19.setMinimumSize(vdim19);
        vkeyboard19.setMaximumSize(vdim19);
        vkeyboard19.setReceiver(recv);

        Dimension vdim12 = new Dimension(1000, 50);
        VirtualKeyboard12 vkeyboard12 = new VirtualKeyboard12();
        vkeyboard12.setSize(vdim12);
        vkeyboard12.setPreferredSize(vdim12);
        vkeyboard12.setMinimumSize(vdim12);
        vkeyboard12.setMaximumSize(vdim12);
        vkeyboard12.setChannel(1);
        vkeyboard12.setReceiver(recv);

        Dimension vdim7 = new Dimension(1700, 50);
        VirtualKeyboard7 vkeyboard7 = new VirtualKeyboard7();
        vkeyboard7.setSize(vdim7);
        vkeyboard7.setPreferredSize(vdim7);
        vkeyboard7.setMinimumSize(vdim7);
        vkeyboard7.setMaximumSize(vdim7);
        vkeyboard7.setChannel(2);
        vkeyboard7.setReceiver(recv);

        Dimension vdim5 = new Dimension(1700, 50);
        VirtualKeyboard5 vkeyboard5 = new VirtualKeyboard5();
        vkeyboard5.setSize(vdim5);
        vkeyboard5.setPreferredSize(vdim5);
        vkeyboard5.setMinimumSize(vdim5);
        vkeyboard5.setMaximumSize(vdim5);
        vkeyboard5.setChannel(3);
        vkeyboard5.setReceiver(recv);

        JScrollPane scrollpane19 = new JScrollPane(vkeyboard19);
        scrollpane19.setPreferredSize(new Dimension(500, 80));
        scrollpane19.getViewport().setViewPosition(new Point(107, 0));

        JScrollPane scrollpane12 = new JScrollPane(vkeyboard12);
        scrollpane12.setPreferredSize(new Dimension(500, 80));
        scrollpane12.getViewport().setViewPosition(new Point(200, 0));

        JScrollPane scrollpane7 = new JScrollPane(vkeyboard7);
        scrollpane7.setPreferredSize(new Dimension(500, 80));
        scrollpane7.getViewport().setViewPosition(new Point(320, 0));

        JScrollPane scrollpane5 = new JScrollPane(vkeyboard5);
        scrollpane5.setPreferredSize(new Dimension(500, 80));
        scrollpane5.getViewport().setViewPosition(new Point(320, 0));

        JPanel panel19 = new JPanel(new BorderLayout());
        panel19.setOpaque(false);
        panel19.setBorder(BorderFactory
                .createTitledBorder("19 equal temperament (19-TET)"));
        panel19.add(scrollpane19);
        boxpanel.add(panel19);

        JPanel panel12 = new JPanel(new BorderLayout());
        panel12.setOpaque(false);
        panel12.setBorder(BorderFactory
                .createTitledBorder("12 equal temperament (12-TET)"));
```

```
247         panel12.add(scrollpane12);
248         boxpanel.add(panel12);
249 
250         JPanel panel7 = new JPanel(new BorderLayout());
251         panel7.setOpaque(false);
252         panel7.setBorder(BorderFactory
253                 .createTitledBorder("7 equal temperament (7-TET)"));
254         panel7.add(scrollpane7);
255         boxpanel.add(panel7);
256 
257         JPanel panel5 = new JPanel(new BorderLayout());
258         panel5.setOpaque(false);
259         panel5.setBorder(BorderFactory
260                 .createTitledBorder("5 equal temperament (5-TET)"));
261         panel5.add(scrollpane5);
262         boxpanel.add(panel5);
263 
264     }
265 
266 }
```

9 TuningApplet2.java

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Point;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;

import javax.sound.midi.InvalidMidiDataException;
import javax.sound.midi.Patch;
import javax.sound.midi.Receiver;
import javax.sound.midi.ShortMessage;
import javax.sound.midi.Soundbank;
import javax.sound.midi.Synthesizer;
import javax.sound.midi.SysexMessage;
import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
import javax.swing.JApplet;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.SwingUtilities;

import com.sun.media.sound.SoftSynthesizer;

public class TuningApplet2 extends JApplet {

    private static final long serialVersionUID = 1L;

    private Synthesizer synth = new SoftSynthesizer();

    private Receiver recv;

    JPanel firstpanel;
    JLabel infolabel;
    String error_text = "";

    public void destroy() {
        synth.close();
    }

    public void init() {
        Runnable runnable = new Runnable() {
            public void run() {
                try {

                    SwingUtilities.invokeAndWait(new Runnable() {
                        public void run() {
                            firstpanel = new JPanel();
                            firstpanel.setBackground(Color.WHITE);
                            firstpanel.setLayout(new FlowLayout());
                            add(firstpanel);

                            infolabel = new JLabel(
                                    "Loading␣synthesizer,␣please␣wait␣.␣.␣.");

                            firstpanel.add(infolabel);

```

```
                        validate();
                        invalidate();
                    }
                });

                synth.getDefaultSoundbank();
                synth.open();
                recv = synth.getReceiver();

                SwingUtilities.invokeAndWait(new Runnable() {
                    public void run() {
                        createGUI();
                        validate();
                        invalidate();
                    }
                });

            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    };
    new Thread(runnable).start();
}

public static void sendTuningChange(Receiver recv, int channel,
        int tuningpreset) throws InvalidMidiDataException {
    // Data Entry
    ShortMessage sm1 = new ShortMessage();
    sm1.setMessage(ShortMessage.CONTROL_CHANGE, channel, 0x64, 03);
    ShortMessage sm2 = new ShortMessage();
    sm2.setMessage(ShortMessage.CONTROL_CHANGE, channel, 0x65, 00);
    // Tuning program 19
    ShortMessage sm3 = new ShortMessage();
    sm3
            .setMessage(ShortMessage.CONTROL_CHANGE, channel, 0x06,
                    tuningpreset);

    // Data Increment
    ShortMessage sm4 = new ShortMessage();
    sm4.setMessage(ShortMessage.CONTROL_CHANGE, channel, 0x60, 0x7F);
    // Data Decrement
    ShortMessage sm5 = new ShortMessage();
    sm5.setMessage(ShortMessage.CONTROL_CHANGE, channel, 0x61, 0x7F);

    recv.send(sm1, -1);
    recv.send(sm2, -1);
    recv.send(sm3, -1);
    recv.send(sm4, -1);
    recv.send(sm5, -1);
}

public static void sendScaleOctaveTunings(Receiver recv, int bank, int preset,
        String name, double[] tunings) throws IOException,
        InvalidMidiDataException {
    int[] itunings = new int[12];
    for (int i = 0; i < itunings.length; i++) {
        itunings[i] = (int) (tunings[i] * 8192.0 / 100.0);
    }
    SysexMessage msg = UniversalSysExBuilder.MidiTuningStandard
            .scaleOctaveTuningDump2ByteForm(UniversalSysExBuilder.ALL_DEVICES, bank, preset,
               name, itunings);
```

```
122         recv.send(msg, -1);
123     }
124 
125     public void createGUI() {
126         remove(firstpanel);
127         JPanel toppanel = new JPanel();
128         toppanel.setBackground(Color.WHITE);
129         toppanel.setLayout(new FlowLayout());
130         add(toppanel);
131 
132         JPanel boxpanel = new JPanel();
133         boxpanel.setOpaque(false);
134         boxpanel.setLayout(new BoxLayout(boxpanel, BoxLayout.Y_AXIS));
135         toppanel.add(boxpanel);
136 
137         Soundbank sbk = synth.getDefaultSoundbank();
138         String[] instruments = new String[128];
139         for (int i = 0; i < instruments.length; i++) {
140             instruments[i] = i + "␣" + sbk.getInstrument(new Patch(0, i)).getName();
141         }
142         final JComboBox instrumentcombobox = new JComboBox(instruments);
143         instrumentcombobox.addActionListener(new ActionListener()
144         {
145             public void actionPerformed(ActionEvent e) {
146                 int pgm = instrumentcombobox.getSelectedIndex();
147                 synth.getChannels()[0].programChange(pgm);
148                 synth.getChannels()[1].programChange(pgm);
149                 synth.getChannels()[2].programChange(pgm);
150                 synth.getChannels()[3].programChange(pgm);
151                 synth.getChannels()[4].programChange(pgm);
152             }
153         });
154         JPanel instrumentpanel = new JPanel();
155         instrumentpanel.setLayout(new FlowLayout(FlowLayout.LEFT));
156         instrumentpanel.setOpaque(false);
157         instrumentpanel.add(new JLabel("Instrument:␣"));
158         instrumentpanel.add(instrumentcombobox);
159         boxpanel.add(instrumentpanel);
160 
161         try {
162 
163             // Just Intonation
164             double[] tunings;
165             tunings = new double[] {
166                     0,
167                     111.73-100.0,
168                     203.91-200.0,
169                     315.64-300.0,
170                     386.31-400.0,
171                     498.04-500.0,
172                     582.51-600.0,
173                     701.96-700.0,
174                     813.69-800.0,
175                     884.36-900.0,
176                     968.826-1000.0,
177                     1088.27-1100.0,
178                     1200.0-1200.0};
179             sendScaleOctaveTunings(recv, 0, 100, "Just", tunings);
180             sendTuningChange(recv, 0, 100);
181 
182             tunings = new double[] {
183                     0,
```

```
                90.22-100.0,
                203.91-200.0,
                294.13-300.0,
                407.82-400.0,
                498.04-500.0,
                611.73-600.0,
                701.96-700.0,
                792.18-800.0,
                905.87-900.0,
                996.09-1000.0,
                1109.78-1100.0};
            sendScaleOctaveTunings(recv, 0, 101, "Pyth", tunings);
            sendTuningChange(recv, 1, 101);

            tunings = new double[] {
                    0.0,
                    76.0-100.0,
                    193.2-200.0,
                    310.3-300.0,
                    386.3-400.0,
                    503.4-500.0,
                    579.5-600.0,
                    696.6-700.0,
                    772.6-800.0,
                    889.7-900.0,
                    1006.8-1000.0,
                    1082.9-1100.0};
            sendScaleOctaveTunings(recv, 0, 102, "Meantone", tunings);
            sendTuningChange(recv, 2, 102);

            tunings = new double[] {
                    0-0,
                    90-100,
                    192-200,
                    294-300,
                    390-400,
                    498-500,
                    588-600,
                    696-700,
                    792-800,
                    888-900,
                    996-1000,
                    1092-1100};
            sendScaleOctaveTunings(recv, 0, 103, "WellTemp", tunings);
            sendTuningChange(recv, 3, 103);



        } catch (Exception e1) {
            e1.printStackTrace();
            return;
        }

        Dimension vdim19 = new Dimension(627, 50);
        VirtualKeyboard19 vkeyboard19 = new VirtualKeyboard19();
        vkeyboard19.setSize(vdim19);
        vkeyboard19.setPreferredSize(vdim19);
        vkeyboard19.setMinimumSize(vdim19);
        vkeyboard19.setMaximumSize(vdim19);
        vkeyboard19.setReceiver(recv);

        for (int i = 0; i < 5; i++) {
```

```
246
247             String title = "";
248             if(i == 0) title = "Just Intonation";
249             if(i == 1) title = "Pythagorean Tuning";
250             if(i == 2) title = "Meantone Temperament";
251             if(i == 3) title = "Well Temperament";
252             if(i == 4) title = "Equal Temperament";
253
254             Dimension vdim12 = new Dimension(1000, 50);
255             VirtualKeyboard12 vkeyboard12 = new VirtualKeyboard12();
256             vkeyboard12.setSize(vdim12);
257             vkeyboard12.setPreferredSize(vdim12);
258             vkeyboard12.setMinimumSize(vdim12);
259             vkeyboard12.setMaximumSize(vdim12);
260             vkeyboard12.setChannel(i);
261             vkeyboard12.setReceiver(recv);
262
263             JScrollPane scrollpane12 = new JScrollPane(vkeyboard12);
264             scrollpane12.setPreferredSize(new Dimension(500, 80));
265             scrollpane12.getViewport().setViewPosition(new Point(200, 0));
266
267             JPanel panel12 = new JPanel(new BorderLayout());
268             panel12.setOpaque(false);
269             panel12.setBorder(BorderFactory
270                     .createTitledBorder(title));
271             panel12.add(scrollpane12);
272             boxpanel.add(panel12);
273         }
274
275     }
276
277 }
```

10 TuningApplet3.java

```
1  import java.awt.BorderLayout;
2  import java.awt.Color;
3  import java.awt.Dimension;
4  import java.awt.FlowLayout;
5  import java.awt.Graphics;
6  import java.awt.Graphics2D;
7  import java.awt.Point;
8  import java.awt.RenderingHints;
9  import java.awt.event.ActionEvent;
10 import java.awt.event.ActionListener;
11 import java.awt.event.MouseEvent;
12 import java.awt.event.MouseListener;
13 import java.awt.event.MouseMotionListener;
14 import java.awt.geom.Rectangle2D;
15 import java.io.IOException;
16
17 import javax.sound.midi.InvalidMidiDataException;
18 import javax.sound.midi.MidiChannel;
19 import javax.sound.midi.Patch;
20 import javax.sound.midi.Receiver;
21 import javax.sound.midi.ShortMessage;
22 import javax.sound.midi.Soundbank;
23 import javax.sound.midi.Synthesizer;
24 import javax.sound.midi.SysexMessage;
25 import javax.swing.BoxLayout;
26 import javax.swing.JApplet;
27 import javax.swing.JComboBox;
28 import javax.swing.JComponent;
29 import javax.swing.JLabel;
30 import javax.swing.JPanel;
31 import javax.swing.JScrollPane;
32 import javax.swing.SwingUtilities;
33
34 import com.sun.media.sound.SoftSynthesizer;
35
36 public class TuningApplet3 extends JApplet {
37
38     private static final long serialVersionUID = 1L;
39
40     private Synthesizer synth = new SoftSynthesizer();
41
42     private Receiver recv;
43
44     JPanel firstpanel;
45
46     JLabel infolabel;
47
48     String error_text = "";
49
50     public void destroy() {
51         synth.close();
52     }
53
54     public static void sendTuningChange(Receiver recv, int channel,
55             int tuningpreset) throws InvalidMidiDataException {
56         // Data Entry
57         ShortMessage sm1 = new ShortMessage();
58         sm1.setMessage(ShortMessage.CONTROL_CHANGE, channel, 0x64, 03);
59         ShortMessage sm2 = new ShortMessage();
60         sm2.setMessage(ShortMessage.CONTROL_CHANGE, channel, 0x65, 00);
```

```
        // Tuning program 19
        ShortMessage sm3 = new ShortMessage();
        sm3
                .setMessage(ShortMessage.CONTROL_CHANGE, channel, 0x06,
                        tuningpreset);

        // Data Increment
        ShortMessage sm4 = new ShortMessage();
        sm4.setMessage(ShortMessage.CONTROL_CHANGE, channel, 0x60, 0x7F);
        // Data Decrement
        ShortMessage sm5 = new ShortMessage();
        sm5.setMessage(ShortMessage.CONTROL_CHANGE, channel, 0x61, 0x7F);

        recv.send(sm1, -1);
        recv.send(sm2, -1);
        recv.send(sm3, -1);
        recv.send(sm4, -1);
        recv.send(sm5, -1);
    }

    public void retune(int bank, int preset, int note, double tuning)
            throws IOException, InvalidMidiDataException {
        int[] key_numbers = new int[1];
        int[] key_tunings = new int[1];

        key_numbers[0] = note;
        key_tunings[0] = (int) (tuning * 16384.0 / 100.0);

        SysexMessage msg = UniversalSysExBuilder.MidiTuningStandard
                .singleNoteTuningChange(UniversalSysExBuilder.ALL_DEVICES,
                        true, bank, preset, key_numbers, key_tunings);
        recv.send(msg, -1);
    }

    public class SoundPainter extends JComponent {

        private static final long serialVersionUID = 1L;

        boolean was_prev_selected = false;

        int was_prev_selected_x;

        int was_prev_selected_y;

        int activenote = -1;

        MidiChannel channel;

        boolean[] activenotes = new boolean[32];

        int[] activenotes_x = new int[32];

        int[] activenotes_y = new int[32];

        public SoundPainter() {
            channel = synth.getChannels()[0];
            addMouseListener(new MouseListener() {

                public void mousePressed(MouseEvent e) {

                    // Check if we clicked on existing note
                    for (int i = 0; i < activenotes.length; i++) {
```

```
123                     if (activenotes[i]) {
124                         was_prev_selected = true;
125                         was_prev_selected_x = e.getX();
126                         was_prev_selected_y = e.getY();
127 
128                         int x_delta = activenotes_x[i] - e.getX();
129                         int y_delta = activenotes_y[i] - e.getY();
130                         int delta = x_delta * x_delta + y_delta * y_delta;
131                         if (delta < 10 * 10) {
132                             // Existing note found
133                             activenote = i;
134                             repaint();
135                             return;
136                         }
137                     }
138                 }
139 
140                 // Find free note
141                 for (int i = 0; i < activenotes.length; i++) {
142                     if (!activenotes[i]) {
143                         was_prev_selected = false;
144                         activenote = i;
145                         activenotes[i] = true;
146                         activenotes_x[i] = e.getX();
147                         activenotes_y[i] = e.getY();
148                         try {
149                             double cent = (e.getX() * 12800.0) / getWidth();
150                             retune(0, 99, 36 + i, cent);
151                         } catch (Exception e1) {
152                             e1.printStackTrace();
153                         }
154                         channel.noteOn(36 + i, 80);
155                         repaint();
156                         return;
157                     }
158                 }
159 
160             }
161 
162             public void mouseReleased(MouseEvent e) {
163 
164                 if (activenote == -1)
165                     return;
166 
167                 if (was_prev_selected) {
168                     int x_delta = was_prev_selected_x - e.getX();
169                     int y_delta = was_prev_selected_y - e.getY();
170                     int delta = x_delta * x_delta + y_delta * y_delta;
171                     if (delta < 3 * 3) {
172                         channel.noteOff(36 + activenote);
173                         activenotes[activenote] = false;
174                         repaint();
175                     }
176                 }
177 
178                 activenote = -1;
179 
180             }
181 
182             public void mouseClicked(MouseEvent e) {
183             }
184 
```

```
185             public void mouseEntered(MouseEvent e) {
186             }
187 
188             public void mouseExited(MouseEvent e) {
189             }
190         });
191 
192         addMouseMotionListener(new MouseMotionListener()
193         {
194             public void mouseDragged(MouseEvent e) {
195                 if (activenote != -1) {
196                     // Existing note found
197                     int i = activenote;
198                     activenotes_x[i] = e.getX();
199                     activenotes_y[i] = e.getY();
200                     try {
201                         double cent = (e.getX() * 12800.0) / getWidth();
202                         retune(0, 99, 36 + i, cent);
203                     } catch (Exception e1) {
204                         e1.printStackTrace();
205                     }
206                     repaint();
207                 }
208             }
209 
210             public void mouseMoved(MouseEvent e) {
211             }
212         });
213     }
214 
215     public void paint(Graphics g) {
216         super.paint(g);
217         Graphics2D g2 = (Graphics2D) g;
218         g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
219                 RenderingHints.VALUE_ANTIALIAS_ON);
220         g2.setRenderingHint(RenderingHints.KEY_FRACTIONALMETRICS,
221                 RenderingHints.VALUE_FRACTIONALMETRICS_ON);
222 
223         g2.setColor(Color.WHITE);
224         g2.fillRect(0, 0, getWidth(), getHeight());
225 
226         int w = getWidth();
227         int h = getHeight();
228         int th = 30;
229 
230         float nw = w / 128f;
231         float cx = 0;
232 
233         g2.setColor(Color.BLACK);
234         g2.drawLine(0, th, w, th);
235 
236         Color light_gray1 = new Color(0.9f, 0.9f, 0.9f);
237         Color light_gray2 = new Color(0.8f, 0.8f, 0.8f);
238 
239         Rectangle2D rect = new Rectangle2D.Double();
240         for (int i = 0; i < 128; i++) {
241             int b = i % 12;
242             boolean a = (b == 1 || b == 3 | b == 6 | b == 8 | b == 10);
243 
244             if (!a) {
245                 g2.setColor(light_gray1);
246                 g2.drawLine((int) cx, th, (int) cx, h);
```

```
247                 } else {
248                     g2.setColor(light_gray2);
249                     g2.drawLine((int) cx, th, (int) cx, h);
250                     g2.drawLine(1 + (int) cx, th, 1 + (int) cx, h);
251                 }
252 
253                 if (!a) {
254                     // rect.setRect(cx, 0, nw, h);
255                     // g2.setColor(Color.WHITE);
256                     // g2.fill(rect);
257 
258                     g2.setColor(Color.BLACK);
259                     // g2.draw(rect);
260                     g2.drawLine((int) cx, 0, (int) cx, th);
261 
262                     // cx += nw;
263 
264                 }
265                 cx += nw;
266             }
267             cx = 0;
268             nw = w / 128f;
269             float black_note_width = nw;
270             for (int i = 0; i < 128; i++) {
271                 int b = i % 12;
272                 boolean a = (b == 1 || b == 3 | b == 6 | b == 8 | b == 10);
273                 if (!a) {
274 
275                 } else {
276                     rect.setRect(cx, th - th * 4.0 / 7.0, black_note_width,
277                             th * 4.0 / 7.0);
278                     g2.setColor(Color.BLACK);
279                     g2.fill(rect);
280                     g2.setColor(Color.BLACK);
281                     g2.draw(rect);
282                 }
283                 cx += nw;
284             }
285 
286             // Draw harmonics
287             for (int j = 0; j < 12; j++) {
288 
289               int ra = j;
290               int rb = j-1;
291               if(j == 0) { ra = 16; rb = 15; }
292               if(j == 1) { ra = 9; rb = 8; }
293               if(j == 2) { ra = 6; rb = 5; }
294               if(j == 3) { ra = 5; rb = 4; }
295               if(j == 4) { ra = 4; rb = 3; }
296               if(j == 5) { ra = 7; rb = 5; }
297               if(j == 6) { ra = 3; rb = 2; }
298               if(j == 7) { ra = 8; rb = 5; }
299               if(j == 8) { ra = 5; rb = 3; }
300               if(j == 9) { ra = 7; rb = 4; }
301               if(j == 10) { ra = 15; rb = 8; }
302               if(j == 11) { ra = 2; rb = 1; }
303 
304               double ratio = ((double)ra) / ((double)rb);
305               double cent = 12.0 * Math.log(ratio) / Math.log(2.0);
306               int jx = (int)(cent * (getWidth() / 128.0));
307               String s_r = ra + "/" + rb;
308 
```

```
309             for (int i = 0; i < activenotes.length; i++) {
310               if (activenotes[i]) {
311                   if (i == activenote)
312                       g2.setColor(Color.BLUE);
313                   else
314                       g2.setColor(Color.LIGHT_GRAY);
315                   g2.fillOval(activenotes_x[i] - 2 + jx, activenotes_y[i] - 2,
316                           4, 4);
317                   if((j % 2) == 0)
318                       g2.drawString(s_r, activenotes_x[i] - 2 + jx, activenotes_y[i] - 3-6);
319                   else
320                       g2.drawString(s_r, activenotes_x[i] - 2 + jx, activenotes_y[i] + 13+6);
321
322                   g2.drawLine(activenotes_x[i]+ jx, activenotes_y[i]-4, activenotes_x[i]+ jx,
                        activenotes_y[i]+3);
323               }
324             }
325           }
326
327           // Draw notes
328           for (int i = 0; i < activenotes.length; i++) {
329               if (activenotes[i]) {
330                   if (i == activenote)
331                       g2.setColor(Color.BLUE);
332                   else
333                       g2.setColor(Color.LIGHT_GRAY);
334                   g2.fillOval(activenotes_x[i] - 10, activenotes_y[i] - 10,
335                           20, 20);
336                   if (i == activenote)
337                   {
338                       g2.drawLine(activenotes_x[i], 0, activenotes_x[i], getHeight());
339                   }
340                   g2.setColor(Color.BLACK);
341                   g2.drawOval(activenotes_x[i] - 1, activenotes_y[i] - 1, 2,
342                           2);
343               }
344           }
345       }
346
347       public void allOff() {
348           for (int i = 0; i < activenotes.length; i++) {
349               if (activenotes[i]) {
350                   channel.noteOff(36 + i);
351                   activenotes[i] = false;
352               }
353           }
354           repaint();
355       }
356   }
357
358   public void init() {
359       Runnable runnable = new Runnable() {
360           public void run() {
361               try {
362
363                   SwingUtilities.invokeAndWait(new Runnable() {
364                       public void run() {
365                           firstpanel = new JPanel();
366                           firstpanel.setBackground(Color.WHITE);
367                           firstpanel.setLayout(new FlowLayout());
368                           add(firstpanel);
369
```

```
                        infolabel = new JLabel(
                                "Loading synthesizer, please wait...");

                        firstpanel.add(infolabel);

                        validate();
                        invalidate();
                    }
                });

                synth.getDefaultSoundbank();
                synth.open();
                recv = synth.getReceiver();

                SwingUtilities.invokeAndWait(new Runnable() {
                    public void run() {
                        createGUI();
                        validate();
                        invalidate();
                    }
                });

            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    };
    new Thread(runnable).start();
}

public void createGUI() {
    remove(firstpanel);
    JPanel toppanel = new JPanel();
    toppanel.setBackground(Color.WHITE);
    toppanel.setLayout(new FlowLayout());
    add(toppanel);

    JPanel boxpanel = new JPanel();
    boxpanel.setOpaque(false);
    boxpanel.setLayout(new BoxLayout(boxpanel, BoxLayout.Y_AXIS));
    toppanel.add(boxpanel);

    final SoundPainter soundpainter = new SoundPainter();

    Soundbank sbk = synth.getDefaultSoundbank();
    String[] instruments = new String[128];
    for (int i = 0; i < instruments.length; i++) {
        instruments[i] = i + " "
                + sbk.getInstrument(new Patch(0, i)).getName();
    }
    final JComboBox instrumentcombobox = new JComboBox(instruments);
    instrumentcombobox.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            int pgm = instrumentcombobox.getSelectedIndex();
            synth.getChannels()[0].programChange(pgm);
            soundpainter.allOff();
        }
    });

    try {
        sendTuningChange(recv, 0, 99);
    } catch (Exception e1) {
```

```
        }
        instrumentcombobox.setSelectedIndex(48);

        JPanel instrumentpanel = new JPanel();
        instrumentpanel.setLayout(new FlowLayout(FlowLayout.LEFT));
        instrumentpanel.setOpaque(false);
        instrumentpanel.add(new JLabel("Instrument: "));
        instrumentpanel.add(instrumentcombobox);
        boxpanel.add(instrumentpanel);

        Dimension vdim12 = new Dimension(3000, 50);
        soundpainter.setSize(vdim12);
        soundpainter.setPreferredSize(vdim12);
        soundpainter.setMinimumSize(vdim12);
        soundpainter.setMaximumSize(vdim12);
        // soundpainter.setChannel(0);
        // soundpainter.setReceiver(recv);

        JScrollPane scrollpane12 = new JScrollPane(soundpainter);
        scrollpane12.setPreferredSize(new Dimension(500, 160));
        scrollpane12.getViewport().setViewPosition(new Point(1200, 0));

        JPanel panel12 = new JPanel(new BorderLayout());
        panel12.setOpaque(false);
        panel12.add(scrollpane12);
        boxpanel.add(panel12);

    }

}
```

11 UniversalSysExBuilder.java

```
1
2
3  import java.io.ByteArrayOutputStream;
4  import java.io.IOException;
5
6  import javax.sound.midi.InvalidMidiDataException;
7  import javax.sound.midi.SysexMessage;
8
9  public class UniversalSysExBuilder {
10
11     public static final int ALL_DEVICES = 0x7F;
12
13     private static final byte[] UNIVERSAL_NON_REALTIME_SYSEX_HEADER = new byte[] {
14             (byte) 0xF0, (byte) 0x7E };
15
16     private static final byte[] UNIVERSAL_REALTIME_SYSEX_HEADER = new byte[] {
17             (byte) 0xF0, (byte) 0x7F };
18
19     private static final byte[] EOX = new byte[] { (byte) 0xF7 };
20
21     public static class GeneralMidiMessages {
22
23         private static final byte[] GENERAL_MIDI_MESSAGES = new byte[] { (byte) 0x09 };
24
25         private static final byte GENERAL_MIDI_1_ON = 0x01;
26
27         private static final byte GENERAL_MIDI_OFF = 0x02;
28
29         private static final byte GENERAL_MIDI_2_ON = 0x03;
30
31         private static SysexMessage setGeneralMidiMessage(int targetDevice,
32                 byte type) throws IOException, InvalidMidiDataException {
33             ByteArrayOutputStream baos = new ByteArrayOutputStream();
34             baos.write(UNIVERSAL_NON_REALTIME_SYSEX_HEADER);
35             baos.write((byte) targetDevice);
36             baos.write(GENERAL_MIDI_MESSAGES);
37             baos.write(type);
38             baos.write(EOX);
39             SysexMessage sysex = new SysexMessage();
40             byte[] data = baos.toByteArray();
41             sysex.setMessage(data, data.length);
42             return sysex;
43
44         }
45
46         public static SysexMessage gmSystemOff(int targetDevice)
47                 throws IOException, InvalidMidiDataException {
48             return setGeneralMidiMessage(targetDevice, GENERAL_MIDI_OFF);
49         }
50
51         public static SysexMessage gmSystemOn(int targetDevice)
52                 throws IOException, InvalidMidiDataException {
53             return setGeneralMidiMessage(targetDevice, GENERAL_MIDI_1_ON);
54         }
55
56         public static SysexMessage gm1SystemOn(int targetDevice)
57                 throws IOException, InvalidMidiDataException {
58             return setGeneralMidiMessage(targetDevice, GENERAL_MIDI_1_ON);
59         }
60
```

```
61          public static SysexMessage gm2SystemOn(int targetDevice)
62                  throws IOException, InvalidMidiDataException {
63              return setGeneralMidiMessage(targetDevice, GENERAL_MIDI_2_ON);
64          }
65      }
66
67      public static class DeviceControl {
68
69          private static final byte[] DEVICE_CONTROL = new byte[] { (byte) 0x04 };
70
71          private static final byte MASTER_VOLUME = (byte) 0x01;
72
73          private static final byte MASTER_BALANCE = (byte) 0x02;
74
75          private static final byte MASTER_FINE_TUNING = (byte) 0x03;
76
77          private static final byte MASTER_COARSE_TUNING = (byte) 0x04;
78
79          private static final byte[] GLOBAL_PARAMETER_CONTROL = new byte[] { (byte) 0x05 };
80
81          private static SysexMessage setDeviceControl(int targetDevice,
82                  int control, int value) throws IOException,
83                  InvalidMidiDataException {
84              ByteArrayOutputStream baos = new ByteArrayOutputStream();
85              baos.write(UNIVERSAL_REALTIME_SYSEX_HEADER);
86              baos.write((byte) targetDevice);
87              baos.write(DEVICE_CONTROL);
88              baos.write((byte) control);
89              baos.write((byte) (value % 128));
90              baos.write((byte) (value / 128));
91              baos.write(EOX);
92              SysexMessage sysex = new SysexMessage();
93              byte[] data = baos.toByteArray();
94              sysex.setMessage(data, data.length);
95              return sysex;
96
97          }
98
99          public static SysexMessage setMasterVolume(int targetDevice, int value)
100                 throws IOException, InvalidMidiDataException {
101             return setDeviceControl(targetDevice, MASTER_VOLUME, value);
102         }
103
104         public static SysexMessage setMasterBalance(int targetDevice, int value)
105                 throws IOException, InvalidMidiDataException {
106             return setDeviceControl(targetDevice, MASTER_BALANCE, value);
107         }
108
109         public static SysexMessage setMasterFineTuning(int targetDevice,
110                 int value) throws IOException, InvalidMidiDataException {
111             return setDeviceControl(targetDevice, MASTER_FINE_TUNING, value);
112         }
113
114         public static SysexMessage setMasterCoarseTuning(int targetDevice,
115                 int value) throws IOException, InvalidMidiDataException {
116             return setDeviceControl(targetDevice, MASTER_COARSE_TUNING, value);
117         }
118
119         public static SysexMessage setGlobalParameter(int targetDevice,
120                 short[] slotpath, byte[] parameter, int value)
121                 throws IOException, InvalidMidiDataException {
122             return setGlobalParameter(targetDevice, slotpath, parameter,
```

```
                new byte[] { (byte) value });
        }

        public static SysexMessage setGlobalParameter(int targetDevice,
                short[] slotpath, byte[] parameter, byte[] value)
                throws IOException, InvalidMidiDataException {
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            baos.write(UNIVERSAL_REALTIME_SYSEX_HEADER);
            baos.write((byte) targetDevice);
            baos.write(DEVICE_CONTROL);
            baos.write(GLOBAL_PARAMETER_CONTROL);
            baos.write((byte) slotpath.length);
            baos.write((byte) parameter.length);
            baos.write((byte) value.length);
            for (int i = 0; i < slotpath.length; i++) {
                short x = slotpath[0];
                baos.write((byte) (x >>> 8));
                baos.write((byte) x);
            }
            baos.write(parameter);
            baos.write(value);
            baos.write(EOX);
            SysexMessage sysex = new SysexMessage();
            byte[] data = baos.toByteArray();
            sysex.setMessage(data, data.length);
            return sysex;
        }

        public static class ReverbEffect {

            public static final int REVERB_TYPE_SMALL_ROOM = 0;

            public static final int REVERB_TYPE_MEDIUM_ROOM = 1;

            public static final int REVERB_TYPE_LARGE_ROOM = 2;

            public static final int REVERB_TYPE_MEDIUM_HALL = 3;

            public static final int REVERB_TYPE_LARGE_HALL = 4;

            public static final int REVERB_TYPE_PLATE = 8;

            private static final short[] SLOTPATH_EFFECT_REVERB = new short[] { (short) 0x0101 };

            private static final byte[] REVERB_TYPE = new byte[] { (byte) 0x00 };

            private static final byte[] REVERB_TIME = new byte[] { (byte) 0x01 };

            public static SysexMessage setReverbType(int targetDevice,
                    int reverbType) throws IOException,
                    InvalidMidiDataException {
                return setGlobalParameter(targetDevice, SLOTPATH_EFFECT_REVERB,
                        REVERB_TYPE, reverbType);
            }

            public static SysexMessage setReverbTime(int targetDevice,
                    int reverbTime) throws IOException,
                    InvalidMidiDataException {
                return setGlobalParameter(targetDevice, SLOTPATH_EFFECT_REVERB,
                        REVERB_TIME, reverbTime);
            }
        }
```

```
185
186         public static class ChorusEffect {
187
188             public static final int CHORUS_TYPE_CHORUS1 = 0;
189
190             public static final int CHORUS_TYPE_CHORUS2 = 1;
191
192             public static final int CHORUS_TYPE_CHORUS3 = 2;
193
194             public static final int CHORUS_TYPE_CHORUS4 = 3;
195
196             public static final int CHORUS_TYPE_FB_CHORUS = 4;
197
198             public static final int CHORUS_TYPE_FLANGER = 5;
199
200             private static final short[] SLOTPATH_EFFECT_CHORUS = new short[] { (short) 0x0102 };
201
202             private static final byte[] CHORUS_TYPE = new byte[] { (byte) 0x00 };
203
204             private static final byte[] CHORUS_MOD_RATE = new byte[] { (byte) 0x01 };
205
206             private static final byte[] CHORUS_MOD_DEPTH = new byte[] { (byte) 0x02 };
207
208             private static final byte[] CHORUS_FEEDBACK = new byte[] { (byte) 0x03 };
209
210             private static final byte[] CHORUS_SEND_TO_REVERB = new byte[] { (byte) 0x04 };
211
212             public static SysexMessage setChorusType(int targetDevice,
213                     int reverbType) throws IOException,
214                     InvalidMidiDataException {
215                 return setGlobalParameter(targetDevice, SLOTPATH_EFFECT_CHORUS,
216                         CHORUS_TYPE, reverbType);
217             }
218
219             public static SysexMessage setChorusModRate(int targetDevice,
220                     int reverbType) throws IOException,
221                     InvalidMidiDataException {
222                 return setGlobalParameter(targetDevice, SLOTPATH_EFFECT_CHORUS,
223                         CHORUS_MOD_RATE, reverbType);
224             }
225
226             public static SysexMessage setChorusModDepth(int targetDevice,
227                     int reverbType) throws IOException,
228                     InvalidMidiDataException {
229                 return setGlobalParameter(targetDevice, SLOTPATH_EFFECT_CHORUS,
230                         CHORUS_MOD_DEPTH, reverbType);
231             }
232
233             public static SysexMessage setChorusFeedback(int targetDevice,
234                     int reverbType) throws IOException,
235                     InvalidMidiDataException {
236                 return setGlobalParameter(targetDevice, SLOTPATH_EFFECT_CHORUS,
237                         CHORUS_FEEDBACK, reverbType);
238             }
239
240             public static SysexMessage setChorusSendToReverb(int targetDevice,
241                     int reverbType) throws IOException,
242                     InvalidMidiDataException {
243                 return setGlobalParameter(targetDevice, SLOTPATH_EFFECT_CHORUS,
244                         CHORUS_SEND_TO_REVERB, reverbType);
245             }
246         }
```

```
247     }
248
249     public static class KeyBasedInstrumentControl {
250
251         public static final int KEY_BASED_CONTORL_FINE_TUNING = 0x78;
252
253         public static final int KEY_BASED_CONTORL_COARSE_TUNING = 0x79;
254
255         private static final byte[] KEY_BASED_INSTRUMENT_CONTROL = new byte[] { (byte) 0x0A };
256
257         private static final byte[] BASIC_MESSAGE = new byte[] { (byte) 0x01 };
258
259         public static SysexMessage setKeyBasedControl(int targetDevice,
260                 int midi_channel, int key_number, int control, int value)
261                 throws IOException, InvalidMidiDataException {
262             return setKeyBasedControl(targetDevice, midi_channel, key_number,
263                     new int[] { control }, new int[] { value });
264         }
265
266         public static SysexMessage setKeyBasedControl(int targetDevice,
267                 int midi_channel, int key_number, int[] controls, int[] values)
268                 throws IOException, InvalidMidiDataException {
269             ByteArrayOutputStream baos = new ByteArrayOutputStream();
270             baos.write(UNIVERSAL_REALTIME_SYSEX_HEADER);
271             baos.write((byte) targetDevice);
272             baos.write(KEY_BASED_INSTRUMENT_CONTROL);
273             baos.write(BASIC_MESSAGE);
274             baos.write((byte) midi_channel);
275             baos.write((byte) key_number);
276             for (int i = 0; i < controls.length; i++) {
277                 baos.write((byte) controls[i]);
278                 baos.write((byte) values[i]);
279             }
280             baos.write(EOX);
281             SysexMessage sysex = new SysexMessage();
282             byte[] data = baos.toByteArray();
283             sysex.setMessage(data, data.length);
284             return sysex;
285         }
286     }
287
288     public static class DestinationSettings {
289
290         private static final byte[] CONTROLLER_DESTINATION_SETTINGS = new byte[] { (byte) 0x09 };
291
292         private static final byte[] CONTROLLER_CHANNEL_PRESSURE = new byte[] { (byte) 0x01 };
293
294         private static final byte[] CONTROLLER_POLY_PRESSURE = new byte[] { (byte) 0x02 };
295
296         private static final byte[] CONTROLLER_CONTROL_CHANGE = new byte[] { (byte) 0x03 };
297
298         public static SysexMessage setControllerDestinationForChannelPressure(
299                 int targetDevice, int channel, int[] controls, int[] ranges)
300                 throws IOException, InvalidMidiDataException {
301             ByteArrayOutputStream baos = new ByteArrayOutputStream();
302             baos.write(UNIVERSAL_REALTIME_SYSEX_HEADER);
303             baos.write((byte) targetDevice);
304             baos.write(CONTROLLER_DESTINATION_SETTINGS);
305             baos.write(CONTROLLER_CHANNEL_PRESSURE);
306             baos.write((byte) channel);
307             for (int i = 0; i < controls.length; i++) {
308                 baos.write((byte) controls[i]);
```

```
309                 baos.write((byte) ranges[i]);
310             }
311             baos.write(EOX);
312             SysexMessage sysex = new SysexMessage();
313             byte[] data = baos.toByteArray();
314             sysex.setMessage(data, data.length);
315             return sysex;
316         }
317 
318         public static SysexMessage setControllerDestinationForPolyPressure(
319                 int targetDevice, int channel, int[] controls, int[] ranges)
320                 throws IOException, InvalidMidiDataException {
321             ByteArrayOutputStream baos = new ByteArrayOutputStream();
322             baos.write(UNIVERSAL_REALTIME_SYSEX_HEADER);
323             baos.write((byte) targetDevice);
324             baos.write(CONTROLLER_DESTINATION_SETTINGS);
325             baos.write(CONTROLLER_POLY_PRESSURE);
326             baos.write((byte) channel);
327             for (int i = 0; i < controls.length; i++) {
328                 baos.write((byte) controls[i]);
329                 baos.write((byte) ranges[i]);
330             }
331             baos.write(EOX);
332             SysexMessage sysex = new SysexMessage();
333             byte[] data = baos.toByteArray();
334             sysex.setMessage(data, data.length);
335             return sysex;
336         }
337 
338         public static SysexMessage setControllerDestinationForControlChange(
339                 int targetDevice, int channel, byte control, int[] controls,
340                 int[] ranges) throws IOException, InvalidMidiDataException {
341             ByteArrayOutputStream baos = new ByteArrayOutputStream();
342             baos.write(UNIVERSAL_REALTIME_SYSEX_HEADER);
343             baos.write((byte) targetDevice);
344             baos.write(CONTROLLER_DESTINATION_SETTINGS);
345             baos.write(CONTROLLER_CONTROL_CHANGE);
346             baos.write((byte) channel);
347             baos.write((byte) control);
348             for (int i = 0; i < controls.length; i++) {
349                 baos.write((byte) controls[i]);
350                 baos.write((byte) ranges[i]);
351             }
352             baos.write(EOX);
353             SysexMessage sysex = new SysexMessage();
354             byte[] data = baos.toByteArray();
355             sysex.setMessage(data, data.length);
356             return sysex;
357         }
358     }
359 
360     public static class MidiTuningStandard {
361 
362         public static final int TUNING_A440 = 45 * 128 * 128;
363 
364         public static final int TUNING_NO_CHANGE = 2097151;
365 
366         private static final byte[] MIDI_TUNING_STANDARD = new byte[] { (byte) 0x08 };
367 
368         private static final byte[] BULK_TUNING_DUMP = new byte[] { (byte) 0x01 };
369 
370         private static final byte[] SINGLE_NOTE_TUNING_CHANGE = new byte[] { (byte) 0x02 };
```

```
371
372         private static final byte[] KEY_BASED_TUNING_DUMP = new byte[] { (byte) 0x04 };
373
374         private static final byte[] SCALE_OCTAVE_TUNING_DUMP_1BYTE_FORM = new byte[] { (byte) 0
                x05 };
375
376         private static final byte[] SCALE_OCTAVE_TUNING_DUMP_2BYTE_FORM = new byte[] { (byte) 0
                x06 };
377
378         private static final byte[] SINGLE_NOTE_TUNING_CHANGE_BANK = new byte[] { (byte) 0x07 };
379
380         private static final byte[] SCALE_OCTAVE_TUNING_1BYTE_FORM = new byte[] { (byte) 0x08 };
381
382         private static final byte[] SCALE_OCTAVE_TUNING_2BYTE_FORM = new byte[] { (byte) 0x09 };
383
384         public static SysexMessage scaleOctaveTuning1ByteForm(int targetDevice,
385                 boolean realtime, boolean channels[], int[] tuning)
386                 throws IOException, InvalidMidiDataException {
387             ByteArrayOutputStream baos = new ByteArrayOutputStream();
388             if (realtime)
389                 baos.write(UNIVERSAL_REALTIME_SYSEX_HEADER);
390             else
391                 baos.write(UNIVERSAL_NON_REALTIME_SYSEX_HEADER);
392             baos.write((byte) targetDevice);
393             baos.write(MIDI_TUNING_STANDARD);
394             baos.write(SCALE_OCTAVE_TUNING_1BYTE_FORM);
395             int channelmask = 0;
396             for (int i = 0; i < 2; i++) {
397                 if (channels[i + 14])
398                     channelmask += 1 << i;
399             }
400             baos.write((byte) channelmask);
401             channelmask = 0;
402             for (int i = 0; i < 7; i++) {
403                 if (channels[i + 7])
404                     channelmask += 1 << i;
405             }
406             baos.write((byte) channelmask);
407             channelmask = 0;
408             for (int i = 0; i < 7; i++) {
409                 if (channels[i])
410                     channelmask += 1 << i;
411             }
412             baos.write((byte) channelmask);
413             for (int i = 0; i < 12; i++) {
414                 baos.write((byte) (tuning[i] + 64));
415             }
416             baos.write(EOX);
417             SysexMessage sysex = new SysexMessage();
418             byte[] data = baos.toByteArray();
419             sysex.setMessage(data, data.length);
420             return sysex;
421         }
422
423         public static SysexMessage scaleOctaveTuning2ByteForm(int targetDevice,
424                 boolean realtime, boolean channels[], int[] tuning)
425                 throws IOException, InvalidMidiDataException {
426             ByteArrayOutputStream baos = new ByteArrayOutputStream();
427             if (realtime)
428                 baos.write(UNIVERSAL_REALTIME_SYSEX_HEADER);
429             else
430                 baos.write(UNIVERSAL_NON_REALTIME_SYSEX_HEADER);
```

```
            baos.write((byte) targetDevice);
            baos.write(MIDI_TUNING_STANDARD);
            baos.write(SCALE_OCTAVE_TUNING_2BYTE_FORM);
            int channelmask = 0;
            for (int i = 0; i < 2; i++) {
                if (channels[i + 14])
                    channelmask += 1 << i;
            }
            baos.write((byte) channelmask);
            channelmask = 0;
            for (int i = 0; i < 7; i++) {
                if (channels[i + 7])
                    channelmask += 1 << i;
            }
            baos.write((byte) channelmask);
            channelmask = 0;
            for (int i = 0; i < 7; i++) {
                if (channels[i])
                    channelmask += 1 << i;
            }
            baos.write((byte) channelmask);
            for (int i = 0; i < 12; i++) {
                int t = tuning[i] + 8192;
                baos.write((byte) (t / 128));
                baos.write((byte) (t % 128));
            }
            baos.write(EOX);
            SysexMessage sysex = new SysexMessage();
            byte[] data = baos.toByteArray();
            sysex.setMessage(data, data.length);
            return sysex;
        }

        private static void setTuningChecksum(byte[] data) {
            int x = data[1] & 0xFF;
            for (int i = 2; i < data.length - 2; i++)
                x = x ^ (data[i] & 0xFF);
            data[data.length - 2] = (byte) (x & 127);
        }

        public static SysexMessage scaleOctaveTuningDump1ByteForm(
                int targetDevice, int bank, int preset, String name,
                int[] tuning) throws IOException, InvalidMidiDataException {
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            baos.write(UNIVERSAL_NON_REALTIME_SYSEX_HEADER);
            baos.write((byte) targetDevice);
            baos.write(MIDI_TUNING_STANDARD);
            baos.write(SCALE_OCTAVE_TUNING_DUMP_1BYTE_FORM);
            baos.write((byte) bank);
            baos.write((byte) preset);
            if (name.length() > 16)
                name = name.substring(0, 16);
            byte[] namebytes = name.getBytes("ASCII");
            baos.write(namebytes);
            byte space_char = " ".getBytes()[0];
            for (int i = namebytes.length; i < 16; i++)
                baos.write(space_char);
            for (int i = 0; i < 12; i++) {
                baos.write((byte) (tuning[i] + 64));
            }
            baos.write(0);
            baos.write(EOX);
```

```
493             SysexMessage sysex = new SysexMessage();
494             byte[] data = baos.toByteArray();
495             setTuningChecksum(data);
496             sysex.setMessage(data, data.length);
497             return sysex;
498         }
499 
500         public static SysexMessage scaleOctaveTuningDump2ByteForm(int targetDevice,
501                 int bank, int preset, String name, int[] tuning)
502                 throws IOException, InvalidMidiDataException {
503             ByteArrayOutputStream baos = new ByteArrayOutputStream();
504             baos.write(UNIVERSAL_NON_REALTIME_SYSEX_HEADER);
505             baos.write((byte) targetDevice);
506             baos.write(MIDI_TUNING_STANDARD);
507             baos.write(SCALE_OCTAVE_TUNING_DUMP_2BYTE_FORM);
508             baos.write((byte) bank);
509             baos.write((byte) preset);
510             if (name.length() > 16)
511                 name = name.substring(0, 16);
512             byte[] namebytes = name.getBytes("ASCII");
513             baos.write(namebytes);
514             byte space_char = "␣".getBytes()[0];
515             for (int i = namebytes.length; i < 16; i++)
516                 baos.write(space_char);
517             for (int i = 0; i < 12; i++) {
518                 int t = tuning[i] + 8192;
519                 baos.write((byte) (t / 128));
520                 baos.write((byte) (t % 128));
521             }
522             baos.write(0);
523             baos.write(EOX);
524             SysexMessage sysex = new SysexMessage();
525             byte[] data = baos.toByteArray();
526             setTuningChecksum(data);
527             sysex.setMessage(data, data.length);
528             return sysex;
529         }
530 
531         public static SysexMessage singleNoteTuningChange(int targetDevice,
532                 boolean realtime, int bank, int preset,
533                 int[] key_numbers, int[] key_tunings) throws IOException,
534                 InvalidMidiDataException {
535             ByteArrayOutputStream baos = new ByteArrayOutputStream();
536             if (realtime)
537                 baos.write(UNIVERSAL_REALTIME_SYSEX_HEADER);
538             else
539                 baos.write(UNIVERSAL_NON_REALTIME_SYSEX_HEADER);
540             baos.write((byte) targetDevice);
541             baos.write(MIDI_TUNING_STANDARD);
542             baos.write(SINGLE_NOTE_TUNING_CHANGE_BANK);
543             baos.write((byte) bank);
544             baos.write((byte) preset);
545             baos.write((byte) key_numbers.length);
546             for (int i = 0; i < key_numbers.length; i++) {
547 
548                 baos.write((byte) key_numbers[i]);
549                 int t = key_tunings[i];
550                 baos.write((byte) ((t / 16384) % 128));
551                 baos.write((byte) ((t / 128) % 128));
552                 baos.write((byte) (t % 128));
553 
554             }
```

```
555             baos.write(EOX);
556             SysexMessage sysex = new SysexMessage();
557             byte[] data = baos.toByteArray();
558             sysex.setMessage(data, data.length);
559             return sysex;
560         }
561 
562         public static SysexMessage singleNoteTuningChange(int targetDevice,
563                 int preset, int[] key_numbers, int[] key_tunings)
564                 throws IOException, InvalidMidiDataException {
565             ByteArrayOutputStream baos = new ByteArrayOutputStream();
566             baos.write(UNIVERSAL_REALTIME_SYSEX_HEADER);
567             baos.write((byte) targetDevice);
568             baos.write(MIDI_TUNING_STANDARD);
569             baos.write(SINGLE_NOTE_TUNING_CHANGE);
570             baos.write((byte) preset);
571             baos.write((byte) key_numbers.length);
572             for (int i = 0; i < key_numbers.length; i++) {
573 
574                 baos.write((byte) key_numbers[i]);
575                 int t = key_tunings[i];
576                 baos.write((byte) ((t / 16384) % 128));
577                 baos.write((byte) ((t / 128) % 128));
578                 baos.write((byte) (t % 128));
579 
580             }
581             baos.write(EOX);
582             SysexMessage sysex = new SysexMessage();
583             byte[] data = baos.toByteArray();
584             sysex.setMessage(data, data.length);
585             return sysex;
586         }
587 
588         public static SysexMessage keyBasedTuningDump(int targetDevice,
589                 int preset, String name, int[] tunings) throws IOException,
590                 InvalidMidiDataException {
591             ByteArrayOutputStream baos = new ByteArrayOutputStream();
592             baos.write(UNIVERSAL_NON_REALTIME_SYSEX_HEADER);
593             baos.write((byte) targetDevice);
594             baos.write(MIDI_TUNING_STANDARD);
595             baos.write(BULK_TUNING_DUMP);
596             baos.write((byte) preset);
597             if (name.length() > 16)
598                 name = name.substring(0, 16);
599             byte[] namebytes = name.getBytes("ASCII");
600             baos.write(namebytes);
601             byte space_char = "␣".getBytes()[0];
602             for (int i = namebytes.length; i < 16; i++)
603                 baos.write(space_char);
604             for (int i = 0; i < 128; i++) {
605                 int t = tunings[i];
606                 baos.write((byte) ((t / 16384) % 128));
607                 baos.write((byte) ((t / 128) % 128));
608                 baos.write((byte) (t % 128));
609             }
610             baos.write(0);
611             baos.write(EOX);
612             SysexMessage sysex = new SysexMessage();
613             byte[] data = baos.toByteArray();
614             setTuningChecksum(data);
615             sysex.setMessage(data, data.length);
616             return sysex;
```

```
617         }
618
619         public static SysexMessage keyBasedTuningDump(int targetDevice,
620                 int bank, int preset, String name, int[] tunings)
621                 throws IOException, InvalidMidiDataException {
622             ByteArrayOutputStream baos = new ByteArrayOutputStream();
623             baos.write(UNIVERSAL_NON_REALTIME_SYSEX_HEADER);
624             baos.write((byte) targetDevice);
625             baos.write(MIDI_TUNING_STANDARD);
626             baos.write(KEY_BASED_TUNING_DUMP);
627             baos.write((byte) bank);
628             baos.write((byte) preset);
629             if (name.length() > 16)
630                 name = name.substring(0, 16);
631             byte[] namebytes = name.getBytes("ASCII");
632             baos.write(namebytes);
633             byte space_char = " ".getBytes()[0];
634             for (int i = namebytes.length; i < 16; i++)
635                 baos.write(space_char);
636             for (int i = 0; i < 128; i++) {
637                 int t = tunings[i];
638                 baos.write((byte) ((t / 16384) % 128));
639                 baos.write((byte) ((t / 128) % 128));
640                 baos.write((byte) (t % 128));
641             }
642             baos.write(0);
643             baos.write(EOX);
644             SysexMessage sysex = new SysexMessage();
645             byte[] data = baos.toByteArray();
646             setTuningChecksum(data);
647             sysex.setMessage(data, data.length);
648             return sysex;
649         }
650
651     }
652
653 }
```

12 VirtualKeyboard12.java

```
1
2
3  import java.awt.Color;
4  import java.awt.Graphics;
5  import java.awt.Graphics2D;
6  import java.awt.Point;
7  import java.awt.RenderingHints;
8  import java.awt.event.FocusEvent;
9  import java.awt.event.FocusListener;
10 import java.awt.event.KeyEvent;
11 import java.awt.event.KeyListener;
12 import java.awt.event.MouseAdapter;
13 import java.awt.event.MouseEvent;
14 import java.awt.geom.Rectangle2D;
15
16 import javax.sound.midi.InvalidMidiDataException;
17 import javax.sound.midi.MidiMessage;
18 import javax.sound.midi.Receiver;
19 import javax.sound.midi.ShortMessage;
20 import javax.sound.midi.Transmitter;
21 import javax.swing.JComponent;
22
23 public class VirtualKeyboard12 extends JComponent implements Receiver, Transmitter {
24
25     private static final long serialVersionUID = 1L;
26
27     private char[] virtualKeys = "zsxdcvgbhnjmq2w3er5t6y7ui9o0p".toCharArray();
28
29     private boolean[] keyDown = new boolean[virtualKeys.length];
30
31     private int lowestKey = 36;
32
33     private Receiver recv = null;
34
35     private int velocity = 80;
36
37     private int channel = 0;
38
39     private boolean[] noteDown = new boolean[128];
40
41     private int midiNoteDown = -1;
42
43     public int getMidiNote(int x, int y)
44     {
45         int w = getWidth();
46         int h = getHeight();
47         float nw = w / 75f;
48
49         int wn = (int)(x / nw);
50         int oct = wn / 7;
51         int n = oct * 12;
52         int nb = wn % 7;
53         if(nb == 1) n += 2;
54         if(nb == 2) n += 4;
55         if(nb == 3) n += 5;
56         if(nb == 4) n += 7;
57         if(nb == 5) n += 9;
58         if(nb == 6) n += 11;
59         if(y < h*4.0/7.0)
60         {
```

```
            int xb = x - (int)(oct * 7 * nw);
            float cx = 0;
            float black_note_width = nw * 0.7f;
            for (int b = 0; b < 12; b++) {
                boolean a = (b==1||b==3|b==6|b==8|b==10);
                if(!a)
                {
                    cx += nw;
                }
                else
                {
                    float cstart = cx - (black_note_width/2);
                    float cend = cstart + black_note_width;
                    if(xb > cstart && xb < cend)
                    {
                        return oct*12 + b;
                    }
                }
            }


        }
        if(n < 0) n = 0;
        if(n > 127) n = 127;
        return n;
    }

    private void allKeyboardKeyOff()
    {
        for (int i = 0; i < keyDown.length; i++) {
            if(keyDown[i])
            if((i + lowestKey) < 128)
            {
                ShortMessage sm = new ShortMessage();
                try {
                    sm.setMessage(ShortMessage.NOTE_OFF, channel, (i + lowestKey), 0);
                    if(recv != null)
                        recv.send(sm, -1);
                    send(sm, -1);
                } catch (InvalidMidiDataException e1) {
                    e1.printStackTrace();
                }
                keyDown[i] = false;
            }
        }
    }

    public void setChannel(int c) {
        channel = c;
    }

    public void setVelocity(int v) {
        velocity = v;
    }

    public VirtualKeyboard12() {
        super();
        setFocusable(true);

        addMouseListener(new MouseAdapter()
        {
            public void mousePressed(MouseEvent e) {
```

```
123                 grabFocus();
124                 Point p = e.getPoint();
125                 midiNoteDown = getMidiNote(p.x, p.y);
126 
127                 ShortMessage sm = new ShortMessage();
128                 try {
129                     sm.setMessage(ShortMessage.NOTE_ON, channel, getMidiNote(p.x, p.y), velocity)
                            ;
130                     if(recv != null)
131                         recv.send(sm, -1);
132                     send(sm, -1);
133                 } catch (InvalidMidiDataException e1) {
134                     e1.printStackTrace();
135                 }
136             }
137 
138             public void mouseReleased(MouseEvent e) {
139                 //Point p = e.getPoint();
140                 //int midiNoteDown = getMidiNote(p.x, p.y);
141                 if(midiNoteDown == -1) return;
142                 ShortMessage sm = new ShortMessage();
143                 try {
144                     sm.setMessage(ShortMessage.NOTE_OFF, channel, midiNoteDown, 0);
145                     if(recv != null)
146                         recv.send(sm, -1);
147                     send(sm, -1);
148                 } catch (InvalidMidiDataException e1) {
149                     e1.printStackTrace();
150                 }
151                 midiNoteDown = -1;
152             }
153         });
154 
155         addKeyListener(new KeyListener()
156         {
157 
158 
159             public void keyPressed(KeyEvent e) {
160                 char lc = Character.toLowerCase(e.getKeyChar());
161                 for (int i = 0; i < virtualKeys.length; i++) {
162                     if(virtualKeys[i] == lc)
163                     {
164                         if(!keyDown[i])
165                         if((i + lowestKey) < 128)
166                         {
167                             ShortMessage sm = new ShortMessage();
168                             try {
169                                 sm.setMessage(ShortMessage.NOTE_ON, channel, (i + lowestKey),
                                        velocity);
170                                 if(recv != null)
171                                     recv.send(sm, -1);
172                                 send(sm, -1);
173                             } catch (InvalidMidiDataException e1) {
174                                 e1.printStackTrace();
175                             }
176                             keyDown[i] = true;
177                         }
178                         return;
179                     }
180                 }
181             }
182 
```

```
183             public void keyReleased(KeyEvent e) {
184                 char lc = Character.toLowerCase(e.getKeyChar());
185                 for (int i = 0; i < virtualKeys.length; i++) {
186                     if(virtualKeys[i] == lc)
187                     {
188                         if(keyDown[i])
189                         if((i + lowestKey) < 128)
190                         {
191                             ShortMessage sm = new ShortMessage();
192                             try {
193                                 sm.setMessage(ShortMessage.NOTE_OFF, channel, (i + lowestKey), 0)
                                        ;
194                                 if(recv != null)
195                                     recv.send(sm, -1);
196                                 send(sm, -1);
197                             } catch (InvalidMidiDataException e1) {
198                                 e1.printStackTrace();
199                             }
200                             keyDown[i] = false;
201                         }
202                         return;
203                     }
204                 }
205             }
206 
207             public void keyTyped(KeyEvent e) {
208 
209                 if(e.getKeyChar() == '-')
210                 {
211                     allKeyboardKeyOff();
212                     lowestKey -= 12;
213                     if(lowestKey < 0) lowestKey = 0;
214                     repaint();
215                 }
216                 if(e.getKeyChar() == '+')
217                 {
218                     allKeyboardKeyOff();
219                     lowestKey += 12;
220                     if(lowestKey > 120) lowestKey = 120;
221                     repaint();
222                 }
223             }
224 
225         });
226 
227         addFocusListener(new FocusListener()
228         {
229             public void focusGained(FocusEvent e) {
230                 repaint();
231             }
232 
233             public void focusLost(FocusEvent e) {
234                 repaint();
235                 allKeyboardKeyOff();
236             }
237 
238         });
239     }
240 
241 
242     public void paint(Graphics g) {
243         super.paint(g);
```

```
        Graphics2D g2 = (Graphics2D)g;
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                RenderingHints.VALUE_ANTIALIAS_ON);
        g2.setRenderingHint(RenderingHints.KEY_FRACTIONALMETRICS,
                RenderingHints.VALUE_FRACTIONALMETRICS_ON);

        int w = getWidth();
        int h = getHeight();

        float nw = w / 75f;
        float cx = 0;
        Rectangle2D rect = new Rectangle2D.Double();
        for (int i = 0; i < 128; i++) {
            int b = i % 12;
            boolean a = (b==1||b==3|b==6|b==8|b==10);
            if(!a)
            {
                rect.setRect(cx, 0, nw, h);
                if(noteDown[i])
                    g2.setColor(new Color(0.8f, 0.8f, 0.95f));
                else
                    g2.setColor(Color.WHITE);
                g2.fill(rect);
                g2.setColor(Color.BLACK);
                g2.draw(rect);

                if(hasFocus() && (i >= lowestKey))
                if(i >= lowestKey)
                {
                    if(i - lowestKey < virtualKeys.length)
                    {
                        g2.setColor(Color.GRAY);
                        char k = virtualKeys[i - lowestKey];
                        g2.drawString("" + k, cx + 2, h - 4);
                    }
                }

                cx += nw;
            }
        }
        cx = 0;
        float black_note_width = nw * 0.7f;
        for (int i = 0; i < 128; i++) {
            int b = i % 12;
            boolean a = (b==1||b==3|b==6|b==8|b==10);
            if(!a)
            {
                cx += nw;
            }
            else
            {
                rect.setRect(cx - (black_note_width/2), 0, black_note_width, h*4.0/7.0);
                if(noteDown[i])
                    g2.setColor(new Color(0.8f, 0.8f, 0.95f));
                else
                    g2.setColor(Color.BLACK);
                g2.fill(rect);
                g2.setColor(Color.BLACK);
                g2.draw(rect);

                if(hasFocus() && (i >= lowestKey))
                    if(i >= lowestKey)
```

```
                    {
                        if(i - lowestKey < virtualKeys.length)
                        {
                            g2.setColor(Color.LIGHT_GRAY);
                            char k = virtualKeys[i - lowestKey];
                            g2.drawString("" + k, cx - (black_note_width/2) + 1, (h*4.0f/7.0f) -
                                3);
                        }
                    }

            }
        }
    }

    public void close() {


    }

    public void send(MidiMessage message, long timeStamp) {
        if(message instanceof ShortMessage)
        {
            ShortMessage sm = (ShortMessage)message;
            if(sm.getChannel() == channel)
            if(sm.getCommand() == ShortMessage.NOTE_ON
                || sm.getCommand() == ShortMessage.NOTE_OFF)
            {
                noteDown[sm.getData1()] =
                    (sm.getCommand() == ShortMessage.NOTE_ON) && (sm.getData2() != 0);
                repaint();
            }
        }
    }

    public Receiver getReceiver() {
        return recv;
    }

    public void setReceiver(Receiver receiver) {
        recv = receiver;
    }

}
```

13 VirtualKeyboard19.java

```
1
2
3  import java.awt.Color;
4  import java.awt.Graphics;
5  import java.awt.Graphics2D;
6  import java.awt.Point;
7  import java.awt.RenderingHints;
8  import java.awt.event.FocusEvent;
9  import java.awt.event.FocusListener;
10 import java.awt.event.KeyEvent;
11 import java.awt.event.KeyListener;
12 import java.awt.event.MouseAdapter;
13 import java.awt.event.MouseEvent;
14 import java.awt.geom.Rectangle2D;
15
16 import javax.sound.midi.InvalidMidiDataException;
17 import javax.sound.midi.MidiMessage;
18 import javax.sound.midi.Receiver;
19 import javax.sound.midi.ShortMessage;
20 import javax.sound.midi.Transmitter;
21 import javax.swing.JComponent;
22
23 public class VirtualKeyboard19 extends JComponent implements Receiver, Transmitter {
24
25     private static final long serialVersionUID = 1L;
26
27     private char[] virtualKeys = "a2ws3edrf5tg6yh7ujik9ol0p".toCharArray();
28
29     // ""
30     private boolean[] keyDown = new boolean[virtualKeys.length];
31
32     private int lowestKey = 38;
33
34     private Receiver recv = null;
35
36     private int velocity = 80;
37
38     private int channel = 0;
39
40     private boolean[] noteDown = new boolean[128];
41
42     private int midiNoteDown = -1;
43
44     public int getMidiNote(int x, int y)
45     {
46         int w = getWidth();
47         int h = getHeight();
48         float nw = w / 47f;
49
50         int wn = (int)(x / nw);
51         int oct = wn / 7;
52         int n = oct * 19;
53         int nb = wn % 7;
54         if(nb == 1) n += 3;
55         if(nb == 2) n += 6;
56         if(nb == 3) n += 8;
57         if(nb == 4) n += 11;
58         if(nb == 5) n += 14;
59         if(nb == 6) n += 17;
60         if(y < h*4.0/7.0)
```

```
61          {
62              int xb = x - (int)(oct * 7 * nw);
63              float cx = 0;
64              float black_note_width = nw * 0.7f;
65              for (int b = 0; b < 19; b++) {
66                  boolean a = !(b==0||b==3||b==6||b==8||b==11||b==14||b==17);
67                  if(!a)
68                  {
69                      cx += nw;
70                  }
71                  else
72                  {
73                      if(b == 7 || b == 18)
74                      {
75                          float cstart = cx - (black_note_width/2);
76                          float cend = cstart + black_note_width;
77                          if(xb > cstart && xb < cend)
78                          {
79                              return oct*19 + b;
80                          }
81                      }
82                      else
83                      {
84                          float cstart = cx - (black_note_width/2);
85                          float cend = cstart + black_note_width;
86                          if(xb > cstart && xb < cend)
87                          {
88                              if(y > (h*4.0/7.0)/2.0)
89                              {
90                                  return oct*19 + b + 1;
91                              }
92                              else
93                                  return oct*19 + b;
94                          }
95                      }
96                  }
97              }
98
99
100         }
101         if(n < 0) n = 0;
102         if(n > 127) n = 127;
103         return n;
104     }
105
106     private void allKeyboardKeyOff()
107     {
108         for (int i = 0; i < keyDown.length; i++) {
109             if(keyDown[i])
110             if((i + lowestKey) < 128)
111             {
112                 ShortMessage sm = new ShortMessage();
113                 try {
114                     sm.setMessage(ShortMessage.NOTE_OFF, channel, (i + lowestKey), 0);
115                     if(recv != null)
116                         recv.send(sm, -1);
117                     send(sm, -1);
118                 } catch (InvalidMidiDataException e1) {
119                     e1.printStackTrace();
120                 }
121                 keyDown[i] = false;
122             }
```

```
123         }
124     }
125 
126     public void setChannel(int c) {
127         channel = c;
128     }
129 
130     public void setVelocity(int v) {
131         velocity = v;
132     }
133 
134     public VirtualKeyboard19() {
135         super();
136         setFocusable(true);
137 
138         addMouseListener(new MouseAdapter()
139         {
140             public void mousePressed(MouseEvent e) {
141                 grabFocus();
142                 Point p = e.getPoint();
143                 midiNoteDown = getMidiNote(p.x, p.y);
144 
145                 ShortMessage sm = new ShortMessage();
146                 try {
147                     sm.setMessage(ShortMessage.NOTE_ON, channel, getMidiNote(p.x, p.y), velocity)
                            ;
148                     if(recv != null)
149                         recv.send(sm, -1);
150                     send(sm, -1);
151                 } catch (InvalidMidiDataException e1) {
152                     e1.printStackTrace();
153                 }
154             }
155 
156             public void mouseReleased(MouseEvent e) {
157                 //Point p = e.getPoint();
158                 //int midiNoteDown = getMidiNote(p.x, p.y);
159                 if(midiNoteDown == -1) return;
160                 ShortMessage sm = new ShortMessage();
161                 try {
162                     sm.setMessage(ShortMessage.NOTE_OFF, channel, midiNoteDown, 0);
163                     if(recv != null)
164                         recv.send(sm, -1);
165                     send(sm, -1);
166                 } catch (InvalidMidiDataException e1) {
167                     e1.printStackTrace();
168                 }
169                 midiNoteDown = -1;
170             }
171         });
172 
173         addKeyListener(new KeyListener()
174         {
175 
176 
177             public void keyPressed(KeyEvent e) {
178                 char lc = Character.toLowerCase(e.getKeyChar());
179                 for (int i = 0; i < virtualKeys.length; i++) {
180                     if(virtualKeys[i] == lc)
181                     {
182                         if(!keyDown[i])
183                         if((i + lowestKey) < 128)
```

```
                    {
                        ShortMessage sm = new ShortMessage();
                        try {
                            sm.setMessage(ShortMessage.NOTE_ON, channel, (i + lowestKey),
                                velocity);
                            if(recv != null)
                                recv.send(sm, -1);
                            send(sm, -1);
                        } catch (InvalidMidiDataException e1) {
                            e1.printStackTrace();
                        }
                        keyDown[i] = true;
                    }
                    return;
                }
            }
        }

        public void keyReleased(KeyEvent e) {
            char lc = Character.toLowerCase(e.getKeyChar());
            for (int i = 0; i < virtualKeys.length; i++) {
                if(virtualKeys[i] == lc)
                {
                    if(keyDown[i])
                    if((i + lowestKey) < 128)
                    {
                        ShortMessage sm = new ShortMessage();
                        try {
                            sm.setMessage(ShortMessage.NOTE_OFF, channel, (i + lowestKey), 0)
                                ;
                            if(recv != null)
                                recv.send(sm, -1);
                            send(sm, -1);
                        } catch (InvalidMidiDataException e1) {
                            e1.printStackTrace();
                        }
                        keyDown[i] = false;
                    }
                    return;
                }
            }
        }

        public void keyTyped(KeyEvent e) {

            if(e.getKeyChar() == '-')
            {
                allKeyboardKeyOff();
                lowestKey -= 19;
                if(lowestKey < 0) lowestKey += 19;
                repaint();
            }
            if(e.getKeyChar() == '+')
            {
                allKeyboardKeyOff();
                lowestKey += 19;
                if(lowestKey > 128) lowestKey -= 19;
                repaint();
            }
        }

    });
```

```
244
245         addFocusListener(new FocusListener()
246         {
247             public void focusGained(FocusEvent e) {
248                 repaint();
249             }
250
251             public void focusLost(FocusEvent e) {
252                 repaint();
253                 allKeyboardKeyOff();
254             }
255
256         });
257     }
258
259
260     public void paint(Graphics g) {
261         super.paint(g);
262         Graphics2D g2 = (Graphics2D)g;
263         g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
264                 RenderingHints.VALUE_ANTIALIAS_ON);
265         g2.setRenderingHint(RenderingHints.KEY_FRACTIONALMETRICS,
266                 RenderingHints.VALUE_FRACTIONALMETRICS_ON);
267
268         int w = getWidth();
269         int h = getHeight();
270
271         float nw = w / 47f;
272         float cx = 0;
273         Rectangle2D rect = new Rectangle2D.Double();
274         for (int i = 0; i < 128; i++) {
275             int b = i % 19;
276             boolean a = !(b==0||b==3||b==6||b==8||b==11||b==14||b==17);
277             if(!a)
278             {
279                 rect.setRect(cx, 0, nw, h);
280                 if(noteDown[i])
281                     g2.setColor(new Color(0.8f, 0.8f, 0.95f));
282                 else
283                     g2.setColor(Color.WHITE);
284                 g2.fill(rect);
285                 g2.setColor(Color.BLACK);
286                 g2.draw(rect);
287
288                 if(hasFocus() && (i >= lowestKey))
289                 if(i >= lowestKey)
290                 {
291                     if(i - lowestKey < virtualKeys.length)
292                     {
293                         g2.setColor(Color.GRAY);
294                         char k = virtualKeys[i - lowestKey];
295                         g2.drawString("" + k, cx + 2, h - 4);
296                     }
297                 }
298
299                 cx += nw;
300             }
301         }
302         cx = 0;
303         float black_note_width = nw * 0.7f;
304         int black_note_pos = 0;
305         for (int i = 0; i < 128; i++) {
```

```
306             int b = i % 19;
307             boolean a = !(b==0||b==3||b==6||b==8||b==11||b==14||b==17);
308             if(!a)
309             {
310                 cx += nw;
311                 black_note_pos = 0;
312             }
313             else
314             {
315                 //7,18
316 
317                 if(b == 7 || b == 18)
318                 {
319                     rect.setRect(cx - (black_note_width/2), 0, black_note_width, h*4.0/7.0);
320                     if(noteDown[i])
321                         g2.setColor(new Color(0.8f, 0.8f, 0.95f));
322                     else
323                         g2.setColor(Color.BLACK);
324                     g2.fill(rect);
325                     g2.setColor(Color.BLACK);
326                     g2.draw(rect);
327 
328                     if(hasFocus() && (i >= lowestKey))
329                         if(i >= lowestKey)
330                         {
331                             if(i - lowestKey < virtualKeys.length)
332                             {
333                                 g2.setColor(Color.LIGHT_GRAY);
334                                 char k = virtualKeys[i - lowestKey];
335                                 g2.drawString("" + k, cx - (black_note_width/2) + 1, (h*4.0f/7.0f
                                        ) - 3);
336                             }
337                         }
338                 }
339                 else
340                 {
341                     if(black_note_pos == 0)
342                     {
343                     rect.setRect(cx - (black_note_width/2), 0, black_note_width, h*4.0/7.0/2-2);
344                     if(noteDown[i])
345                         g2.setColor(new Color(0.8f, 0.8f, 0.95f));
346                     else
347                         g2.setColor(Color.BLACK);
348                     g2.fill(rect);
349                     g2.setColor(Color.BLACK);
350                     g2.draw(rect);
351 
352                     if(hasFocus() && (i >= lowestKey))
353                         if(i >= lowestKey)
354                         {
355                             if(i - lowestKey < virtualKeys.length)
356                             {
357                                 g2.setColor(Color.LIGHT_GRAY);
358                                 char k = virtualKeys[i - lowestKey];
359                                 g2.drawString("" + k, cx - (black_note_width/2) + 1, (h*2.0f/7.0f
                                        ) - 5);
360                             }
361                         }
362                     }
363 
364                     if(black_note_pos == 1)
365                     {
```

```
366                     rect.setRect(cx - (black_note_width/2), h*4.0/7.0/2+1, black_note_width, h
                            *4.0/7.0/2-1);
367                     if(noteDown[i])
368                         g2.setColor(new Color(0.8f, 0.8f, 0.95f));
369                     else
370                         g2.setColor(Color.BLACK);
371                     g2.fill(rect);
372                     g2.setColor(Color.BLACK);
373                     g2.draw(rect);
374 
375                     if(hasFocus() && (i >= lowestKey))
376                         if(i >= lowestKey)
377                         {
378                             if(i - lowestKey < virtualKeys.length)
379                             {
380                                 g2.setColor(Color.LIGHT_GRAY);
381                                 char k = virtualKeys[i - lowestKey];
382                                 g2.drawString("" + k, cx - (black_note_width/2) + 1, (h*4.0f/7.0f
                                        ) - 3);
383                             }
384                         }
385                     }
386 
387                 }
388 
389                 black_note_pos ++;
390 
391             }
392         }
393     }
394 
395     public void close() {
396 
397 
398     }
399 
400     public void send(MidiMessage message, long timeStamp) {
401         if(message instanceof ShortMessage)
402         {
403             ShortMessage sm = (ShortMessage)message;
404             if(sm.getChannel() == channel)
405             if(sm.getCommand() == ShortMessage.NOTE_ON
406                 || sm.getCommand() == ShortMessage.NOTE_OFF)
407             {
408                 noteDown[sm.getData1()] =
409                     (sm.getCommand() == ShortMessage.NOTE_ON) && (sm.getData2() != 0);
410                 repaint();
411             }
412         }
413     }
414 
415     public Receiver getReceiver() {
416         return recv;
417     }
418 
419     public void setReceiver(Receiver receiver) {
420         recv = receiver;
421     }
422 
423 }
```

14 VirtualKeyboard5.java

```
1
2
3  import java.awt.Color;
4  import java.awt.Graphics;
5  import java.awt.Graphics2D;
6  import java.awt.Point;
7  import java.awt.RenderingHints;
8  import java.awt.event.FocusEvent;
9  import java.awt.event.FocusListener;
10 import java.awt.event.KeyEvent;
11 import java.awt.event.KeyListener;
12 import java.awt.event.MouseAdapter;
13 import java.awt.event.MouseEvent;
14 import java.awt.geom.Rectangle2D;
15
16 import javax.sound.midi.InvalidMidiDataException;
17 import javax.sound.midi.MidiMessage;
18 import javax.sound.midi.Receiver;
19 import javax.sound.midi.ShortMessage;
20 import javax.sound.midi.Transmitter;
21 import javax.swing.JComponent;
22
23 public class VirtualKeyboard5 extends JComponent implements Receiver, Transmitter {
24
25     private static final long serialVersionUID = 1L;
26
27     private char[] virtualKeys = "zxcvbasdfgqwertyuiop1234567890".toCharArray();
28
29     private boolean[] keyDown = new boolean[virtualKeys.length];
30
31     private int lowestKey = 25;
32
33     private Receiver recv = null;
34
35     private int velocity = 80;
36
37     private int channel = 0;
38
39     private boolean[] noteDown = new boolean[128];
40
41     private int midiNoteDown = -1;
42
43     public int getMidiNote(int x, int y)
44     {
45         int w = getWidth();
46         float nw = w / 128f;
47
48         int wn = (int)(x / nw);
49         int oct = wn / 7;
50         int n = oct * 7 + wn % 7;
51         if(n < 0) n = 0;
52         if(n > 127) n = 127;
53         return n;
54     }
55
56     private void allKeyboardKeyOff()
57     {
58         for (int i = 0; i < keyDown.length; i++) {
59             if(keyDown[i])
60             if((i + lowestKey) < 128)
```

```
61              {
62                  ShortMessage sm = new ShortMessage();
63                  try {
64                      sm.setMessage(ShortMessage.NOTE_OFF, channel, (i + lowestKey), 0);
65                      if(recv != null)
66                          recv.send(sm, -1);
67                      send(sm, -1);
68                  } catch (InvalidMidiDataException e1) {
69                      e1.printStackTrace();
70                  }
71                  keyDown[i] = false;
72              }
73          }
74      }
75
76      public void setChannel(int c) {
77          channel = c;
78      }
79
80      public void setVelocity(int v) {
81          velocity = v;
82      }
83
84      public VirtualKeyboard5() {
85          super();
86          setFocusable(true);
87
88          addMouseListener(new MouseAdapter()
89          {
90              public void mousePressed(MouseEvent e) {
91                  grabFocus();
92                  Point p = e.getPoint();
93                  midiNoteDown = getMidiNote(p.x, p.y);
94
95                  ShortMessage sm = new ShortMessage();
96                  try {
97                      sm.setMessage(ShortMessage.NOTE_ON, channel, getMidiNote(p.x, p.y), velocity)
                            ;
98                      if(recv != null)
99                          recv.send(sm, -1);
100                     send(sm, -1);
101                 } catch (InvalidMidiDataException e1) {
102                     e1.printStackTrace();
103                 }
104             }
105
106             public void mouseReleased(MouseEvent e) {
107                 //Point p = e.getPoint();
108                 //int midiNoteDown = getMidiNote(p.x, p.y);
109                 if(midiNoteDown == -1) return;
110                 ShortMessage sm = new ShortMessage();
111                 try {
112                     sm.setMessage(ShortMessage.NOTE_OFF, channel, midiNoteDown, 0);
113                     if(recv != null)
114                         recv.send(sm, -1);
115                     send(sm, -1);
116                 } catch (InvalidMidiDataException e1) {
117                     e1.printStackTrace();
118                 }
119                 midiNoteDown = -1;
120             }
121         });
```

```
122
123         addKeyListener(new KeyListener()
124         {
125
126
127             public void keyPressed(KeyEvent e) {
128                 char lc = Character.toLowerCase(e.getKeyChar());
129                 for (int i = 0; i < virtualKeys.length; i++) {
130                     if(virtualKeys[i] == lc)
131                     {
132                         if(!keyDown[i])
133                         if((i + lowestKey) < 128)
134                         {
135                             ShortMessage sm = new ShortMessage();
136                             try {
137                                 sm.setMessage(ShortMessage.NOTE_ON, channel, (i + lowestKey),
                                        velocity);
138                                 if(recv != null)
139                                     recv.send(sm, -1);
140                                 send(sm, -1);
141                             } catch (InvalidMidiDataException e1) {
142                                 e1.printStackTrace();
143                             }
144                             keyDown[i] = true;
145                         }
146                         return;
147                     }
148                 }
149             }
150
151             public void keyReleased(KeyEvent e) {
152                 char lc = Character.toLowerCase(e.getKeyChar());
153                 for (int i = 0; i < virtualKeys.length; i++) {
154                     if(virtualKeys[i] == lc)
155                     {
156                         if(keyDown[i])
157                         if((i + lowestKey) < 128)
158                         {
159                             ShortMessage sm = new ShortMessage();
160                             try {
161                                 sm.setMessage(ShortMessage.NOTE_OFF, channel, (i + lowestKey), 0)
                                        ;
162                                 if(recv != null)
163                                     recv.send(sm, -1);
164                                 send(sm, -1);
165                             } catch (InvalidMidiDataException e1) {
166                                 e1.printStackTrace();
167                             }
168                             keyDown[i] = false;
169                         }
170                         return;
171                     }
172                 }
173             }
174
175             public void keyTyped(KeyEvent e) {
176
177                 if(e.getKeyChar() == '-')
178                 {
179                     allKeyboardKeyOff();
180                     lowestKey -= 7;
181                     if(lowestKey < 0) lowestKey = 0;
```

```
                repaint();
            }
            if(e.getKeyChar() == '+')
            {
                allKeyboardKeyOff();
                lowestKey += 7;
                if(lowestKey > 127) lowestKey -= 7;
                repaint();
            }
        }

    });

    addFocusListener(new FocusListener()
    {
        public void focusGained(FocusEvent e) {
            repaint();
        }

        public void focusLost(FocusEvent e) {
            repaint();
            allKeyboardKeyOff();
        }

    });
}


public void paint(Graphics g) {
    super.paint(g);
    Graphics2D g2 = (Graphics2D)g;
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);
    g2.setRenderingHint(RenderingHints.KEY_FRACTIONALMETRICS,
            RenderingHints.VALUE_FRACTIONALMETRICS_ON);

    int w = getWidth();
    int h = getHeight();

    float nw = w / 128f;
    float cx = 0;
    Rectangle2D rect = new Rectangle2D.Double();
    for (int i = 0; i < 128; i++) {

            rect.setRect(cx, 0, nw, h);
            if(noteDown[i])
                g2.setColor(new Color(0.8f, 0.8f, 0.95f));
            else
                g2.setColor(Color.WHITE);
            g2.fill(rect);
            g2.setColor(Color.BLACK);
            g2.draw(rect);

            if(i % 5 == 0)
            g2.drawString("C", cx + 2, 12);

            if(hasFocus() && (i >= lowestKey))
            if(i >= lowestKey)
            {
                if(i - lowestKey < virtualKeys.length)
                {
                    g2.setColor(Color.GRAY);
```

```
                        char k = virtualKeys[i - lowestKey];
                        g2.drawString("" + k, cx + 2, h - 4);
                    }
                }

                cx += nw;
            }
        }

    public void close() {


    }

    public void send(MidiMessage message, long timeStamp) {
        if(message instanceof ShortMessage)
        {
            ShortMessage sm = (ShortMessage)message;
            if(sm.getChannel() == channel)
            if(sm.getCommand() == ShortMessage.NOTE_ON
                || sm.getCommand() == ShortMessage.NOTE_OFF)
            {
                noteDown[sm.getData1()] =
                    (sm.getCommand() == ShortMessage.NOTE_ON) && (sm.getData2() != 0);
                repaint();
            }
        }
    }

    public Receiver getReceiver() {
        return recv;
    }

    public void setReceiver(Receiver receiver) {
        recv = receiver;
    }

}
```

15 VirtualKeyboard7.java

```
1
2
3  import java.awt.Color;
4  import java.awt.Graphics;
5  import java.awt.Graphics2D;
6  import java.awt.Point;
7  import java.awt.RenderingHints;
8  import java.awt.event.FocusEvent;
9  import java.awt.event.FocusListener;
10 import java.awt.event.KeyEvent;
11 import java.awt.event.KeyListener;
12 import java.awt.event.MouseAdapter;
13 import java.awt.event.MouseEvent;
14 import java.awt.geom.Rectangle2D;
15
16 import javax.sound.midi.InvalidMidiDataException;
17 import javax.sound.midi.MidiMessage;
18 import javax.sound.midi.Receiver;
19 import javax.sound.midi.ShortMessage;
20 import javax.sound.midi.Transmitter;
21 import javax.swing.JComponent;
22
23 public class VirtualKeyboard7 extends JComponent implements Receiver, Transmitter {
24
25     private static final long serialVersionUID = 1L;
26
27     private char[] virtualKeys = "zxcvbnmasdfghjqwertyu1234567".toCharArray();
28
29     private boolean[] keyDown = new boolean[virtualKeys.length];
30
31     private int lowestKey = 35;
32
33     private Receiver recv = null;
34
35     private int velocity = 80;
36
37     private int channel = 0;
38
39     private boolean[] noteDown = new boolean[128];
40
41     private int midiNoteDown = -1;
42
43     public int getMidiNote(int x, int y)
44     {
45         int w = getWidth();
46         float nw = w / 128f;
47
48         int wn = (int)(x / nw);
49         int oct = wn / 7;
50         int n = oct * 7 + wn % 7;
51         if(n < 0) n = 0;
52         if(n > 127) n = 127;
53         return n;
54     }
55
56     private void allKeyboardKeyOff()
57     {
58         for (int i = 0; i < keyDown.length; i++) {
59             if(keyDown[i])
60             if((i + lowestKey) < 128)
```

```
61              {
62                  ShortMessage sm = new ShortMessage();
63                  try {
64                      sm.setMessage(ShortMessage.NOTE_OFF, channel, (i + lowestKey), 0);
65                      if(recv != null)
66                          recv.send(sm, -1);
67                      send(sm, -1);
68                  } catch (InvalidMidiDataException e1) {
69                      e1.printStackTrace();
70                  }
71                  keyDown[i] = false;
72              }
73          }
74      }
75
76      public void setChannel(int c) {
77          channel = c;
78      }
79
80      public void setVelocity(int v) {
81          velocity = v;
82      }
83
84      public VirtualKeyboard7() {
85          super();
86          setFocusable(true);
87
88          addMouseListener(new MouseAdapter()
89          {
90              public void mousePressed(MouseEvent e) {
91                  grabFocus();
92                  Point p = e.getPoint();
93                  midiNoteDown = getMidiNote(p.x, p.y);
94
95                  ShortMessage sm = new ShortMessage();
96                  try {
97                      sm.setMessage(ShortMessage.NOTE_ON, channel, getMidiNote(p.x, p.y), velocity)
                            ;
98                      if(recv != null)
99                          recv.send(sm, -1);
100                     send(sm, -1);
101                 } catch (InvalidMidiDataException e1) {
102                     e1.printStackTrace();
103                 }
104             }
105
106             public void mouseReleased(MouseEvent e) {
107                 //Point p = e.getPoint();
108                 //int midiNoteDown = getMidiNote(p.x, p.y);
109                 if(midiNoteDown == -1) return;
110                 ShortMessage sm = new ShortMessage();
111                 try {
112                     sm.setMessage(ShortMessage.NOTE_OFF, channel, midiNoteDown, 0);
113                     if(recv != null)
114                         recv.send(sm, -1);
115                     send(sm, -1);
116                 } catch (InvalidMidiDataException e1) {
117                     e1.printStackTrace();
118                 }
119                 midiNoteDown = -1;
120             }
121         });
```

```
122
123         addKeyListener(new KeyListener()
124         {
125
126
127             public void keyPressed(KeyEvent e) {
128                 char lc = Character.toLowerCase(e.getKeyChar());
129                 for (int i = 0; i < virtualKeys.length; i++) {
130                     if(virtualKeys[i] == lc)
131                     {
132                         if(!keyDown[i])
133                         if((i + lowestKey) < 128)
134                         {
135                             ShortMessage sm = new ShortMessage();
136                             try {
137                                 sm.setMessage(ShortMessage.NOTE_ON, channel, (i + lowestKey),
                                        velocity);
138                                 if(recv != null)
139                                     recv.send(sm, -1);
140                                 send(sm, -1);
141                             } catch (InvalidMidiDataException e1) {
142                                 e1.printStackTrace();
143                             }
144                             keyDown[i] = true;
145                         }
146                         return;
147                     }
148                 }
149             }
150
151             public void keyReleased(KeyEvent e) {
152                 char lc = Character.toLowerCase(e.getKeyChar());
153                 for (int i = 0; i < virtualKeys.length; i++) {
154                     if(virtualKeys[i] == lc)
155                     {
156                         if(keyDown[i])
157                         if((i + lowestKey) < 128)
158                         {
159                             ShortMessage sm = new ShortMessage();
160                             try {
161                                 sm.setMessage(ShortMessage.NOTE_OFF, channel, (i + lowestKey), 0)
                                        ;
162                                 if(recv != null)
163                                     recv.send(sm, -1);
164                                 send(sm, -1);
165                             } catch (InvalidMidiDataException e1) {
166                                 e1.printStackTrace();
167                             }
168                             keyDown[i] = false;
169                         }
170                         return;
171                     }
172                 }
173             }
174
175             public void keyTyped(KeyEvent e) {
176
177                 if(e.getKeyChar() == '-')
178                 {
179                     allKeyboardKeyOff();
180                     lowestKey -= 7;
181                     if(lowestKey < 0) lowestKey = 0;
```

```
                repaint();
            }
            if(e.getKeyChar() == '+')
            {
                allKeyboardKeyOff();
                lowestKey += 7;
                if(lowestKey > 127) lowestKey -= 7;
                repaint();
            }
        }

    });

    addFocusListener(new FocusListener()
    {
        public void focusGained(FocusEvent e) {
            repaint();
        }

        public void focusLost(FocusEvent e) {
            repaint();
            allKeyboardKeyOff();
        }

    });
}


public void paint(Graphics g) {
    super.paint(g);
    Graphics2D g2 = (Graphics2D)g;
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);
    g2.setRenderingHint(RenderingHints.KEY_FRACTIONALMETRICS,
            RenderingHints.VALUE_FRACTIONALMETRICS_ON);

    int w = getWidth();
    int h = getHeight();

    float nw = w / 128f;
    float cx = 0;
    Rectangle2D rect = new Rectangle2D.Double();
    for (int i = 0; i < 128; i++) {

            rect.setRect(cx, 0, nw, h);
            if(noteDown[i])
                g2.setColor(new Color(0.8f, 0.8f, 0.95f));
            else
                g2.setColor(Color.WHITE);
            g2.fill(rect);
            g2.setColor(Color.BLACK);
            g2.draw(rect);

            if(i % 7 == 0)
            g2.drawString("C", cx + 2, 12);

            if(hasFocus() && (i >= lowestKey))
            if(i >= lowestKey)
            {
                if(i - lowestKey < virtualKeys.length)
                {
                    g2.setColor(Color.GRAY);
```

```
                        char k = virtualKeys[i - lowestKey];
                        g2.drawString("" + k, cx + 2, h - 4);
                    }
                }

                cx += nw;
            }
        }

    public void close() {


    }

    public void send(MidiMessage message, long timeStamp) {
        if(message instanceof ShortMessage)
        {
            ShortMessage sm = (ShortMessage)message;
            if(sm.getChannel() == channel)
            if(sm.getCommand() == ShortMessage.NOTE_ON
                || sm.getCommand() == ShortMessage.NOTE_OFF)
            {
                noteDown[sm.getData1()] =
                    (sm.getCommand() == ShortMessage.NOTE_ON) && (sm.getData2() != 0);
                repaint();
            }
        }
    }

    public Receiver getReceiver() {
        return recv;
    }

    public void setReceiver(Receiver receiver) {
        recv = receiver;
    }

}
```

16 com/sun/media/sound/AudioFileSoundbankReader.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.io.ByteArrayOutputStream;
import java.io.DataInputStream;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;

import javax.sound.midi.InvalidMidiDataException;
import javax.sound.midi.Soundbank;
import javax.sound.midi.spi.SoundbankReader;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.UnsupportedAudioFileException;

/**
 * Soundbank reader that uses audio files as soundbanks.
 *
 * @author Karl Helgason
 */
public class AudioFileSoundbankReader extends SoundbankReader {

    public Soundbank getSoundbank(URL url)
            throws InvalidMidiDataException, IOException {
        try {
            AudioInputStream ais = AudioSystem.getAudioInputStream(url);
            Soundbank sbk = getSoundbank(ais);
            ais.close();
            return sbk;
        } catch (UnsupportedAudioFileException e) {
            return null;
        } catch (IOException e) {
            return null;
        }
    }
```

```
61
62      public Soundbank getSoundbank(InputStream stream)
63              throws InvalidMidiDataException, IOException {
64          stream.mark(512);
65          try {
66              AudioInputStream ais = AudioSystem.getAudioInputStream(stream);
67              Soundbank sbk = getSoundbank(ais);
68              if (sbk != null)
69                  return sbk;
70          } catch (UnsupportedAudioFileException e) {
71          } catch (IOException e) {
72          }
73          stream.reset();
74          return null;
75      }
76
77      public Soundbank getSoundbank(AudioInputStream ais)
78              throws InvalidMidiDataException, IOException {
79          try {
80              byte[] buffer;
81              if (ais.getFrameLength() == -1) {
82                  ByteArrayOutputStream baos = new ByteArrayOutputStream();
83                  byte[] buff = new byte[1024
84                          - (1024 % ais.getFormat().getFrameSize())];
85                  int ret;
86                  while ((ret = ais.read(buff)) != -1) {
87                      baos.write(buff, 0, ret);
88                  }
89                  ais.close();
90                  buffer = baos.toByteArray();
91              } else {
92                  buffer = new byte[(int) (ais.getFrameLength()
93                                  * ais.getFormat().getFrameSize())];
94                  new DataInputStream(ais).readFully(buffer);
95              }
96              ModelByteBufferWavetable osc = new ModelByteBufferWavetable(
97                      new ModelByteBuffer(buffer), ais.getFormat(), -4800);
98              ModelPerformer performer = new ModelPerformer();
99              performer.getOscillators().add(osc);
100
101             SimpleSoundbank sbk = new SimpleSoundbank();
102             SimpleInstrument ins = new SimpleInstrument();
103             ins.add(performer);
104             sbk.addInstrument(ins);
105             return sbk;
106         } catch (Exception e) {
107             return null;
108         }
109     }
110
111     public Soundbank getSoundbank(File file)
112             throws InvalidMidiDataException, IOException {
113         try {
114             AudioInputStream ais = AudioSystem.getAudioInputStream(file);
115             ais.close();
116             ModelByteBufferWavetable osc = new ModelByteBufferWavetable(
117                     new ModelByteBuffer(file, 0, file.length()), -4800);
118             ModelPerformer performer = new ModelPerformer();
119             performer.getOscillators().add(osc);
120             SimpleSoundbank sbk = new SimpleSoundbank();
121             SimpleInstrument ins = new SimpleInstrument();
122             ins.add(performer);
```

```
123             sbk.addInstrument(ins);
124             return sbk;
125         } catch (UnsupportedAudioFileException e1) {
126             return null;
127         } catch (IOException e) {
128             return null;
129         }
130     }
131 }
```

17 com/sun/media/sound/AudioFloatConverter.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.DoubleBuffer;
import java.nio.FloatBuffer;

import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioFormat.Encoding;

/**
 * This class is used to convert between 8,16,24,32,32+ bit signed/unsigned
 * big/litle endian fixed/floating point byte buffers and float buffers.
 *
 * @author Karl Helgason
 */
public abstract class AudioFloatConverter {

    public static final Encoding PCM_FLOAT = new Encoding("PCM_FLOAT");

    /**************************************************************************
     *
     * LSB Filter, used filter least significant byte in samples arrays.
     *
     * Is used filter out data in lsb byte when SampleSizeInBits is not
     * dividable by 8.
     *
     *************************************************************************/

    private static class AudioFloatLSBFilter extends AudioFloatConverter {

        private AudioFloatConverter converter;

        final private int offset;

        final private int stepsize;
```

```
61
62          final private byte mask;
63
64          private byte[] mask_buffer;
65
66          public AudioFloatLSBFilter(AudioFloatConverter converter,
67                  AudioFormat format) {
68              int bits = format.getSampleSizeInBits();
69              boolean bigEndian = format.isBigEndian();
70              this.converter = converter;
71              stepsize = (bits + 7) / 8;
72              offset = bigEndian ? (stepsize - 1) : 0;
73              int lsb_bits = bits % 8;
74              if (lsb_bits == 0)
75                  mask = (byte) 0x00;
76              else if (lsb_bits == 1)
77                  mask = (byte) 0x80;
78              else if (lsb_bits == 2)
79                  mask = (byte) 0xC0;
80              else if (lsb_bits == 3)
81                  mask = (byte) 0xE0;
82              else if (lsb_bits == 4)
83                  mask = (byte) 0xF0;
84              else if (lsb_bits == 5)
85                  mask = (byte) 0xF8;
86              else if (lsb_bits == 6)
87                  mask = (byte) 0xFC;
88              else if (lsb_bits == 7)
89                  mask = (byte) 0xFE;
90              else
91                  mask = (byte) 0xFF;
92          }
93
94          public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
95                  byte[] out_buff, int out_offset) {
96              byte[] ret = converter.toByteArray(in_buff, in_offset, in_len,
97                      out_buff, out_offset);
98
99              int out_offset_end = in_len * stepsize;
100             for (int i = out_offset + offset; i < out_offset_end; i += stepsize) {
101                 out_buff[i] = (byte) (out_buff[i] & mask);
102             }
103
104             return ret;
105         }
106
107         public float[] toFloatArray(byte[] in_buff, int in_offset,
108                 float[] out_buff, int out_offset, int out_len) {
109             if (mask_buffer == null || mask_buffer.length < in_buff.length)
110                 mask_buffer = new byte[in_buff.length];
111             System.arraycopy(in_buff, 0, mask_buffer, 0, in_buff.length);
112             int in_offset_end = out_len * stepsize;
113             for (int i = in_offset + offset; i < in_offset_end; i += stepsize) {
114                 mask_buffer[i] = (byte) (mask_buffer[i] & mask);
115             }
116             float[] ret = converter.toFloatArray(mask_buffer, in_offset,
117                     out_buff, out_offset, out_len);
118             return ret;
119         }
120
121     }
122
```

```
    /**************************************************************************
     *
     * 64 bit float, little/big-endian
     *
     *************************************************************************/

    // PCM 64 bit float, little-endian
    private static class AudioFloatConversion64L extends AudioFloatConverter {
        ByteBuffer bytebuffer = null;

        DoubleBuffer floatbuffer = null;

        double[] double_buff = null;

        public float[] toFloatArray(byte[] in_buff, int in_offset,
                float[] out_buff, int out_offset, int out_len) {
            int in_len = out_len * 8;
            if (bytebuffer == null || bytebuffer.capacity() < in_len) {
                bytebuffer = ByteBuffer.allocate(in_len).order(
                        ByteOrder.LITTLE_ENDIAN);
                floatbuffer = bytebuffer.asDoubleBuffer();
            }
            bytebuffer.position(0);
            floatbuffer.position(0);
            bytebuffer.put(in_buff, in_offset, in_len);
            if (double_buff == null
                    || double_buff.length < out_len + out_offset)
                double_buff = new double[out_len + out_offset];
            floatbuffer.get(double_buff, out_offset, out_len);
            int out_offset_end = out_offset + out_len;
            for (int i = out_offset; i < out_offset_end; i++) {
                out_buff[i] = (float) double_buff[i];
            }
            return out_buff;
        }

        public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
                byte[] out_buff, int out_offset) {
            int out_len = in_len * 8;
            if (bytebuffer == null || bytebuffer.capacity() < out_len) {
                bytebuffer = ByteBuffer.allocate(out_len).order(
                        ByteOrder.LITTLE_ENDIAN);
                floatbuffer = bytebuffer.asDoubleBuffer();
            }
            floatbuffer.position(0);
            bytebuffer.position(0);
            if (double_buff == null || double_buff.length < in_offset + in_len)
                double_buff = new double[in_offset + in_len];
            int in_offset_end = in_offset + in_len;
            for (int i = in_offset; i < in_offset_end; i++) {
                double_buff[i] = in_buff[i];
            }
            floatbuffer.put(double_buff, in_offset, in_len);
            bytebuffer.get(out_buff, out_offset, out_len);
            return out_buff;
        }
    }

    // PCM 64 bit float, big-endian
    private static class AudioFloatConversion64B extends AudioFloatConverter {
        ByteBuffer bytebuffer = null;

```

```
        DoubleBuffer floatbuffer = null;

        double[] double_buff = null;

        public float[] toFloatArray(byte[] in_buff, int in_offset,
                float[] out_buff, int out_offset, int out_len) {
            int in_len = out_len * 8;
            if (bytebuffer == null || bytebuffer.capacity() < in_len) {
                bytebuffer = ByteBuffer.allocate(in_len).order(
                        ByteOrder.BIG_ENDIAN);
                floatbuffer = bytebuffer.asDoubleBuffer();
            }
            bytebuffer.position(0);
            floatbuffer.position(0);
            bytebuffer.put(in_buff, in_offset, in_len);
            if (double_buff == null
                    || double_buff.length < out_len + out_offset)
                double_buff = new double[out_len + out_offset];
            floatbuffer.get(double_buff, out_offset, out_len);
            int out_offset_end = out_offset + out_len;
            for (int i = out_offset; i < out_offset_end; i++) {
                out_buff[i] = (float) double_buff[i];
            }
            return out_buff;
        }

        public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
                byte[] out_buff, int out_offset) {
            int out_len = in_len * 8;
            if (bytebuffer == null || bytebuffer.capacity() < out_len) {
                bytebuffer = ByteBuffer.allocate(out_len).order(
                        ByteOrder.BIG_ENDIAN);
                floatbuffer = bytebuffer.asDoubleBuffer();
            }
            floatbuffer.position(0);
            bytebuffer.position(0);
            if (double_buff == null || double_buff.length < in_offset + in_len)
                double_buff = new double[in_offset + in_len];
            int in_offset_end = in_offset + in_len;
            for (int i = in_offset; i < in_offset_end; i++) {
                double_buff[i] = in_buff[i];
            }
            floatbuffer.put(double_buff, in_offset, in_len);
            bytebuffer.get(out_buff, out_offset, out_len);
            return out_buff;
        }
    }

    /**********************************************************************
     *
     * 32 bit float, little/big-endian
     *
     **********************************************************************/

    // PCM 32 bit float, little-endian
    private static class AudioFloatConversion32L extends AudioFloatConverter {
        ByteBuffer bytebuffer = null;

        FloatBuffer floatbuffer = null;

        public float[] toFloatArray(byte[] in_buff, int in_offset,
                float[] out_buff, int out_offset, int out_len) {
```

```
            int in_len = out_len * 4;
            if (bytebuffer == null || bytebuffer.capacity() < in_len) {
                bytebuffer = ByteBuffer.allocate(in_len).order(
                        ByteOrder.LITTLE_ENDIAN);
                floatbuffer = bytebuffer.asFloatBuffer();
            }
            bytebuffer.position(0);
            floatbuffer.position(0);
            bytebuffer.put(in_buff, in_offset, in_len);
            floatbuffer.get(out_buff, out_offset, out_len);
            return out_buff;
        }

        public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
                byte[] out_buff, int out_offset) {
            int out_len = in_len * 4;
            if (bytebuffer == null || bytebuffer.capacity() < out_len) {
                bytebuffer = ByteBuffer.allocate(out_len).order(
                        ByteOrder.LITTLE_ENDIAN);
                floatbuffer = bytebuffer.asFloatBuffer();
            }
            floatbuffer.position(0);
            bytebuffer.position(0);
            floatbuffer.put(in_buff, in_offset, in_len);
            bytebuffer.get(out_buff, out_offset, out_len);
            return out_buff;
        }
    }

    // PCM 32 bit float, big-endian
    private static class AudioFloatConversion32B extends AudioFloatConverter {
        ByteBuffer bytebuffer = null;

        FloatBuffer floatbuffer = null;

        public float[] toFloatArray(byte[] in_buff, int in_offset,
                float[] out_buff, int out_offset, int out_len) {
            int in_len = out_len * 4;
            if (bytebuffer == null || bytebuffer.capacity() < in_len) {
                bytebuffer = ByteBuffer.allocate(in_len).order(
                        ByteOrder.BIG_ENDIAN);
                floatbuffer = bytebuffer.asFloatBuffer();
            }
            bytebuffer.position(0);
            floatbuffer.position(0);
            bytebuffer.put(in_buff, in_offset, in_len);
            floatbuffer.get(out_buff, out_offset, out_len);
            return out_buff;
        }

        public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
                byte[] out_buff, int out_offset) {
            int out_len = in_len * 4;
            if (bytebuffer == null || bytebuffer.capacity() < out_len) {
                bytebuffer = ByteBuffer.allocate(out_len).order(
                        ByteOrder.BIG_ENDIAN);
                floatbuffer = bytebuffer.asFloatBuffer();
            }
            floatbuffer.position(0);
            bytebuffer.position(0);
            floatbuffer.put(in_buff, in_offset, in_len);
            bytebuffer.get(out_buff, out_offset, out_len);
```

```
309             return out_buff;
310         }
311     }
312 
313     /**************************************************************************
314      *
315      * 8 bit signed/unsigned
316      *
317      *************************************************************************/
318 
319     // PCM 8 bit, signed
320     private static class AudioFloatConversion8S extends AudioFloatConverter {
321         public float[] toFloatArray(byte[] in_buff, int in_offset,
322                 float[] out_buff, int out_offset, int out_len) {
323             int ix = in_offset;
324             int ox = out_offset;
325             for (int i = 0; i < out_len; i++)
326                 out_buff[ox++] = in_buff[ix++] * (1.0f / 127.0f);
327             return out_buff;
328         }
329 
330         public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
331                 byte[] out_buff, int out_offset) {
332             int ix = in_offset;
333             int ox = out_offset;
334             for (int i = 0; i < in_len; i++)
335                 out_buff[ox++] = (byte) (in_buff[ix++] * 127.0f);
336             return out_buff;
337         }
338     }
339 
340     // PCM 8 bit, unsigned
341     private static class AudioFloatConversion8U extends AudioFloatConverter {
342         public float[] toFloatArray(byte[] in_buff, int in_offset,
343                 float[] out_buff, int out_offset, int out_len) {
344             int ix = in_offset;
345             int ox = out_offset;
346             for (int i = 0; i < out_len; i++)
347                 out_buff[ox++] = ((in_buff[ix++] & 0xFF) - 127)
348                         * (1.0f / 127.0f);
349             return out_buff;
350         }
351 
352         public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
353                 byte[] out_buff, int out_offset) {
354             int ix = in_offset;
355             int ox = out_offset;
356             for (int i = 0; i < in_len; i++)
357                 out_buff[ox++] = (byte) (127 + in_buff[ix++] * 127.0f);
358             return out_buff;
359         }
360     }
361 
362     /**************************************************************************
363      *
364      * 16 bit signed/unsigned, little/big-endian
365      *
366      *************************************************************************/
367 
368     // PCM 16 bit, signed, little-endian
369     private static class AudioFloatConversion16SL extends AudioFloatConverter {
370         public float[] toFloatArray(byte[] in_buff, int in_offset,
```

```
                float[] out_buff, int out_offset, int out_len) {
            int ix = in_offset;
            int len = out_offset + out_len;
            for (int ox = out_offset; ox < len; ox++) {
                out_buff[ox] = ((short) ((in_buff[ix++] & 0xFF) |
                           (in_buff[ix++] << 8))) * (1.0f / 32767.0f);
            }

            return out_buff;
        }

        public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
                byte[] out_buff, int out_offset) {
            int ox = out_offset;
            int len = in_offset + in_len;
            for (int ix = in_offset; ix < len; ix++) {
                int x = (int) (in_buff[ix] * 32767.0);
                out_buff[ox++] = (byte) x;
                out_buff[ox++] = (byte) (x >>> 8);
            }
            return out_buff;
        }
    }

    // PCM 16 bit, signed, big-endian
    private static class AudioFloatConversion16SB extends AudioFloatConverter {
        public float[] toFloatArray(byte[] in_buff, int in_offset,
                float[] out_buff, int out_offset, int out_len) {
            int ix = in_offset;
            int ox = out_offset;
            for (int i = 0; i < out_len; i++) {
                out_buff[ox++] = ((short) ((in_buff[ix++] << 8) |
                        (in_buff[ix++] & 0xFF))) * (1.0f / 32767.0f);
            }
            return out_buff;
        }

        public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
                byte[] out_buff, int out_offset) {
            int ix = in_offset;
            int ox = out_offset;
            for (int i = 0; i < in_len; i++) {
                int x = (int) (in_buff[ix++] * 32767.0);
                out_buff[ox++] = (byte) (x >>> 8);
                out_buff[ox++] = (byte) x;
            }
            return out_buff;
        }
    }

    // PCM 16 bit, unsigned, little-endian
    private static class AudioFloatConversion16UL extends AudioFloatConverter {
        public float[] toFloatArray(byte[] in_buff, int in_offset,
                float[] out_buff, int out_offset, int out_len) {
            int ix = in_offset;
            int ox = out_offset;
            for (int i = 0; i < out_len; i++) {
                int x = (in_buff[ix++] & 0xFF) | ((in_buff[ix++] & 0xFF) << 8);
                out_buff[ox++] = (x - 32767) * (1.0f / 32767.0f);
            }
            return out_buff;
        }
```

```
433
434         public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
435                 byte[] out_buff, int out_offset) {
436             int ix = in_offset;
437             int ox = out_offset;
438             for (int i = 0; i < in_len; i++) {
439                 int x = 32767 + (int) (in_buff[ix++] * 32767.0);
440                 out_buff[ox++] = (byte) x;
441                 out_buff[ox++] = (byte) (x >>> 8);
442             }
443             return out_buff;
444         }
445     }
446
447     // PCM 16 bit, unsigned, big-endian
448     private static class AudioFloatConversion16UB extends AudioFloatConverter {
449         public float[] toFloatArray(byte[] in_buff, int in_offset,
450                 float[] out_buff, int out_offset, int out_len) {
451             int ix = in_offset;
452             int ox = out_offset;
453             for (int i = 0; i < out_len; i++) {
454                 int x = ((in_buff[ix++] & 0xFF) << 8) | (in_buff[ix++] & 0xFF);
455                 out_buff[ox++] = (x - 32767) * (1.0f / 32767.0f);
456             }
457             return out_buff;
458         }
459
460         public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
461                 byte[] out_buff, int out_offset) {
462             int ix = in_offset;
463             int ox = out_offset;
464             for (int i = 0; i < in_len; i++) {
465                 int x = 32767 + (int) (in_buff[ix++] * 32767.0);
466                 out_buff[ox++] = (byte) (x >>> 8);
467                 out_buff[ox++] = (byte) x;
468             }
469             return out_buff;
470         }
471     }
472
473     /**************************************************************************
474      *
475      * 24 bit signed/unsigned, little/big-endian
476      *
477      **************************************************************************/
478
479     // PCM 24 bit, signed, little-endian
480     private static class AudioFloatConversion24SL extends AudioFloatConverter {
481         public float[] toFloatArray(byte[] in_buff, int in_offset,
482                 float[] out_buff, int out_offset, int out_len) {
483             int ix = in_offset;
484             int ox = out_offset;
485             for (int i = 0; i < out_len; i++) {
486                 int x = (in_buff[ix++] & 0xFF) | ((in_buff[ix++] & 0xFF) << 8)
487                         | ((in_buff[ix++] & 0xFF) << 16);
488                 if (x > 0x7FFFFF)
489                     x -= 0x1000000;
490                 out_buff[ox++] = x * (1.0f / (float)0x7FFFFF);
491             }
492             return out_buff;
493         }
494
```

```
        public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
                byte[] out_buff, int out_offset) {
            int ix = in_offset;
            int ox = out_offset;
            for (int i = 0; i < in_len; i++) {
                int x = (int) (in_buff[ix++] * (float)0x7FFFFF);
                if (x < 0)
                    x += 0x1000000;
                out_buff[ox++] = (byte) x;
                out_buff[ox++] = (byte) (x >>> 8);
                out_buff[ox++] = (byte) (x >>> 16);
            }
            return out_buff;
        }
    }

    // PCM 24 bit, signed, big-endian
    private static class AudioFloatConversion24SB extends AudioFloatConverter {
        public float[] toFloatArray(byte[] in_buff, int in_offset,
                float[] out_buff, int out_offset, int out_len) {
            int ix = in_offset;
            int ox = out_offset;
            for (int i = 0; i < out_len; i++) {
                int x = ((in_buff[ix++] & 0xFF) << 16)
                        | ((in_buff[ix++] & 0xFF) << 8) | (in_buff[ix++] & 0xFF);
                if (x > 0x7FFFFF)
                    x -= 0x1000000;
                out_buff[ox++] = x * (1.0f / (float)0x7FFFFF);
            }
            return out_buff;
        }

        public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
                byte[] out_buff, int out_offset) {
            int ix = in_offset;
            int ox = out_offset;
            for (int i = 0; i < in_len; i++) {
                int x = (int) (in_buff[ix++] * (float)0x7FFFFF);
                if (x < 0)
                    x += 0x1000000;
                out_buff[ox++] = (byte) (x >>> 16);
                out_buff[ox++] = (byte) (x >>> 8);
                out_buff[ox++] = (byte) x;
            }
            return out_buff;
        }
    }

    // PCM 24 bit, unsigned, little-endian
    private static class AudioFloatConversion24UL extends AudioFloatConverter {
        public float[] toFloatArray(byte[] in_buff, int in_offset,
                float[] out_buff, int out_offset, int out_len) {
            int ix = in_offset;
            int ox = out_offset;
            for (int i = 0; i < out_len; i++) {
                int x = (in_buff[ix++] & 0xFF) | ((in_buff[ix++] & 0xFF) << 8)
                        | ((in_buff[ix++] & 0xFF) << 16);
                x -= 0x7FFFFF;
                out_buff[ox++] = x * (1.0f / (float)0x7FFFFF);
            }
            return out_buff;
        }
```

```
557
558         public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
559                 byte[] out_buff, int out_offset) {
560             int ix = in_offset;
561             int ox = out_offset;
562             for (int i = 0; i < in_len; i++) {
563                 int x = (int) (in_buff[ix++] * (float)0x7FFFFF);
564                 x += 0x7FFFFF;
565                 out_buff[ox++] = (byte) x;
566                 out_buff[ox++] = (byte) (x >>> 8);
567                 out_buff[ox++] = (byte) (x >>> 16);
568             }
569             return out_buff;
570         }
571     }
572
573     // PCM 24 bit, unsigned, big-endian
574     private static class AudioFloatConversion24UB extends AudioFloatConverter {
575         public float[] toFloatArray(byte[] in_buff, int in_offset,
576                 float[] out_buff, int out_offset, int out_len) {
577             int ix = in_offset;
578             int ox = out_offset;
579             for (int i = 0; i < out_len; i++) {
580                 int x = ((in_buff[ix++] & 0xFF) << 16)
581                         | ((in_buff[ix++] & 0xFF) << 8) | (in_buff[ix++] & 0xFF);
582                 x -= 0x7FFFFF;
583                 out_buff[ox++] = x * (1.0f / (float)0x7FFFFF);
584             }
585             return out_buff;
586         }
587
588         public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
589                 byte[] out_buff, int out_offset) {
590             int ix = in_offset;
591             int ox = out_offset;
592             for (int i = 0; i < in_len; i++) {
593                 int x = (int) (in_buff[ix++] * (float)0x7FFFFF);
594                 x += 0x7FFFFF;
595                 out_buff[ox++] = (byte) (x >>> 16);
596                 out_buff[ox++] = (byte) (x >>> 8);
597                 out_buff[ox++] = (byte) x;
598             }
599             return out_buff;
600         }
601     }
602
603     /**************************************************************************
604      *
605      * 32 bit signed/unsigned, little/big-endian
606      *
607      **************************************************************************/
608
609     // PCM 32 bit, signed, little-endian
610     private static class AudioFloatConversion32SL extends AudioFloatConverter {
611         public float[] toFloatArray(byte[] in_buff, int in_offset,
612                 float[] out_buff, int out_offset, int out_len) {
613             int ix = in_offset;
614             int ox = out_offset;
615             for (int i = 0; i < out_len; i++) {
616                 int x = (in_buff[ix++] & 0xFF) | ((in_buff[ix++] & 0xFF) << 8) |
617                         ((in_buff[ix++] & 0xFF) << 16) |
618                         ((in_buff[ix++] & 0xFF) << 24);
```

```
                out_buff[ox++] = x * (1.0f / (float)0x7FFFFFFF);
            }
            return out_buff;
        }

        public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
                byte[] out_buff, int out_offset) {
            int ix = in_offset;
            int ox = out_offset;
            for (int i = 0; i < in_len; i++) {
                int x = (int) (in_buff[ix++] * (float)0x7FFFFFFF);
                out_buff[ox++] = (byte) x;
                out_buff[ox++] = (byte) (x >>> 8);
                out_buff[ox++] = (byte) (x >>> 16);
                out_buff[ox++] = (byte) (x >>> 24);
            }
            return out_buff;
        }
    }

    // PCM 32 bit, signed, big-endian
    private static class AudioFloatConversion32SB extends AudioFloatConverter {
        public float[] toFloatArray(byte[] in_buff, int in_offset,
                float[] out_buff, int out_offset, int out_len) {
            int ix = in_offset;
            int ox = out_offset;
            for (int i = 0; i < out_len; i++) {
                int x = ((in_buff[ix++] & 0xFF) << 24) |
                        ((in_buff[ix++] & 0xFF) << 16) |
                        ((in_buff[ix++] & 0xFF) << 8) | (in_buff[ix++] & 0xFF);
                out_buff[ox++] = x * (1.0f / (float)0x7FFFFFFF);
            }
            return out_buff;
        }

        public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
                byte[] out_buff, int out_offset) {
            int ix = in_offset;
            int ox = out_offset;
            for (int i = 0; i < in_len; i++) {
                int x = (int) (in_buff[ix++] * (float)0x7FFFFFFF);
                out_buff[ox++] = (byte) (x >>> 24);
                out_buff[ox++] = (byte) (x >>> 16);
                out_buff[ox++] = (byte) (x >>> 8);
                out_buff[ox++] = (byte) x;
            }
            return out_buff;
        }
    }

    // PCM 32 bit, unsigned, little-endian
    private static class AudioFloatConversion32UL extends AudioFloatConverter {
        public float[] toFloatArray(byte[] in_buff, int in_offset,
                float[] out_buff, int out_offset, int out_len) {
            int ix = in_offset;
            int ox = out_offset;
            for (int i = 0; i < out_len; i++) {
                int x = (in_buff[ix++] & 0xFF) | ((in_buff[ix++] & 0xFF) << 8) |
                        ((in_buff[ix++] & 0xFF) << 16) |
                        ((in_buff[ix++] & 0xFF) << 24);
                x -= 0x7FFFFFFF;
                out_buff[ox++] = x * (1.0f / (float)0x7FFFFFFF);
```

```
            }
            return out_buff;
        }

        public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
                byte[] out_buff, int out_offset) {
            int ix = in_offset;
            int ox = out_offset;
            for (int i = 0; i < in_len; i++) {
                int x = (int) (in_buff[ix++] * (float)0x7FFFFFFF);
                x += 0x7FFFFFFF;
                out_buff[ox++] = (byte) x;
                out_buff[ox++] = (byte) (x >>> 8);
                out_buff[ox++] = (byte) (x >>> 16);
                out_buff[ox++] = (byte) (x >>> 24);
            }
            return out_buff;
        }
    }

    // PCM 32 bit, unsigned, big-endian
    private static class AudioFloatConversion32UB extends AudioFloatConverter {

        public float[] toFloatArray(byte[] in_buff, int in_offset,
                float[] out_buff, int out_offset, int out_len) {
            int ix = in_offset;
            int ox = out_offset;
            for (int i = 0; i < out_len; i++) {
                int x = ((in_buff[ix++] & 0xFF) << 24) |
                        ((in_buff[ix++] & 0xFF) << 16) |
                        ((in_buff[ix++] & 0xFF) << 8) | (in_buff[ix++] & 0xFF);
                x -= 0x7FFFFFFF;
                out_buff[ox++] = x * (1.0f / (float)0x7FFFFFFF);
            }
            return out_buff;
        }

        public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
                byte[] out_buff, int out_offset) {
            int ix = in_offset;
            int ox = out_offset;
            for (int i = 0; i < in_len; i++) {
                int x = (int) (in_buff[ix++] * (float)0x7FFFFFFF);
                x += 0x7FFFFFFF;
                out_buff[ox++] = (byte) (x >>> 24);
                out_buff[ox++] = (byte) (x >>> 16);
                out_buff[ox++] = (byte) (x >>> 8);
                out_buff[ox++] = (byte) x;
            }
            return out_buff;
        }
    }

    /**************************************************************************
     *
     * 32+ bit signed/unsigned, little/big-endian
     *
     **************************************************************************/

    // PCM 32+ bit, signed, little-endian
    private static class AudioFloatConversion32xSL extends AudioFloatConverter {

```

```
        final int xbytes;

        public AudioFloatConversion32xSL(int xbytes) {
            this.xbytes = xbytes;
        }

        public float[] toFloatArray(byte[] in_buff, int in_offset,
                float[] out_buff, int out_offset, int out_len) {
            int ix = in_offset;
            int ox = out_offset;
            for (int i = 0; i < out_len; i++) {
                ix += xbytes;
                int x = (in_buff[ix++] & 0xFF) | ((in_buff[ix++] & 0xFF) << 8)
                        | ((in_buff[ix++] & 0xFF) << 16)
                        | ((in_buff[ix++] & 0xFF) << 24);
                out_buff[ox++] = x * (1.0f / (float)0x7FFFFFFF);
            }
            return out_buff;
        }

        public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
                byte[] out_buff, int out_offset) {
            int ix = in_offset;
            int ox = out_offset;
            for (int i = 0; i < in_len; i++) {
                int x = (int) (in_buff[ix++] * (float)0x7FFFFFFF);
                for (int j = 0; j < xbytes; j++) {
                    out_buff[ox++] = 0;
                }
                out_buff[ox++] = (byte) x;
                out_buff[ox++] = (byte) (x >>> 8);
                out_buff[ox++] = (byte) (x >>> 16);
                out_buff[ox++] = (byte) (x >>> 24);
            }
            return out_buff;
        }
    }

    // PCM 32+ bit, signed, big-endian
    private static class AudioFloatConversion32xSB extends AudioFloatConverter {

        final int xbytes;

        public AudioFloatConversion32xSB(int xbytes) {
            this.xbytes = xbytes;
        }

        public float[] toFloatArray(byte[] in_buff, int in_offset,
                float[] out_buff, int out_offset, int out_len) {
            int ix = in_offset;
            int ox = out_offset;
            for (int i = 0; i < out_len; i++) {
                int x = ((in_buff[ix++] & 0xFF) << 24)
                        | ((in_buff[ix++] & 0xFF) << 16)
                        | ((in_buff[ix++] & 0xFF) << 8)
                        | (in_buff[ix++] & 0xFF);
                ix += xbytes;
                out_buff[ox++] = x * (1.0f / (float)0x7FFFFFFF);
            }
            return out_buff;
        }

```

```
        public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
                byte[] out_buff, int out_offset) {
            int ix = in_offset;
            int ox = out_offset;
            for (int i = 0; i < in_len; i++) {
                int x = (int) (in_buff[ix++] * (float)0x7FFFFFFF);
                out_buff[ox++] = (byte) (x >>> 24);
                out_buff[ox++] = (byte) (x >>> 16);
                out_buff[ox++] = (byte) (x >>> 8);
                out_buff[ox++] = (byte) x;
                for (int j = 0; j < xbytes; j++) {
                    out_buff[ox++] = 0;
                }
            }
            return out_buff;
        }
    }

    // PCM 32+ bit, unsigned, little-endian
    private static class AudioFloatConversion32xUL extends AudioFloatConverter {

        final int xbytes;

        public AudioFloatConversion32xUL(int xbytes) {
            this.xbytes = xbytes;
        }

        public float[] toFloatArray(byte[] in_buff, int in_offset,
                float[] out_buff, int out_offset, int out_len) {
            int ix = in_offset;
            int ox = out_offset;
            for (int i = 0; i < out_len; i++) {
                ix += xbytes;
                int x = (in_buff[ix++] & 0xFF) | ((in_buff[ix++] & 0xFF) << 8)
                        | ((in_buff[ix++] & 0xFF) << 16)
                        | ((in_buff[ix++] & 0xFF) << 24);
                x -= 0x7FFFFFFF;
                out_buff[ox++] = x * (1.0f / (float)0x7FFFFFFF);
            }
            return out_buff;
        }

        public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
                byte[] out_buff, int out_offset) {
            int ix = in_offset;
            int ox = out_offset;
            for (int i = 0; i < in_len; i++) {
                int x = (int) (in_buff[ix++] * (float)0x7FFFFFFF);
                x += 0x7FFFFFFF;
                for (int j = 0; j < xbytes; j++) {
                    out_buff[ox++] = 0;
                }
                out_buff[ox++] = (byte) x;
                out_buff[ox++] = (byte) (x >>> 8);
                out_buff[ox++] = (byte) (x >>> 16);
                out_buff[ox++] = (byte) (x >>> 24);
            }
            return out_buff;
        }
    }

    // PCM 32+ bit, unsigned, big-endian
```

```
867     private static class AudioFloatConversion32xUB extends AudioFloatConverter {
868
869         final int xbytes;
870
871         public AudioFloatConversion32xUB(int xbytes) {
872             this.xbytes = xbytes;
873         }
874
875         public float[] toFloatArray(byte[] in_buff, int in_offset,
876                 float[] out_buff, int out_offset, int out_len) {
877             int ix = in_offset;
878             int ox = out_offset;
879             for (int i = 0; i < out_len; i++) {
880                 int x = ((in_buff[ix++] & 0xFF) << 24) |
881                         ((in_buff[ix++] & 0xFF) << 16) |
882                         ((in_buff[ix++] & 0xFF) << 8) | (in_buff[ix++] & 0xFF);
883                 ix += xbytes;
884                 x -= 2147483647;
885                 out_buff[ox++] = x * (1.0f / 2147483647.0f);
886             }
887             return out_buff;
888         }
889
890         public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
891                 byte[] out_buff, int out_offset) {
892             int ix = in_offset;
893             int ox = out_offset;
894             for (int i = 0; i < in_len; i++) {
895                 int x = (int) (in_buff[ix++] * 2147483647.0);
896                 x += 2147483647;
897                 out_buff[ox++] = (byte) (x >>> 24);
898                 out_buff[ox++] = (byte) (x >>> 16);
899                 out_buff[ox++] = (byte) (x >>> 8);
900                 out_buff[ox++] = (byte) x;
901                 for (int j = 0; j < xbytes; j++) {
902                     out_buff[ox++] = 0;
903                 }
904             }
905             return out_buff;
906         }
907     }
908
909     public static AudioFloatConverter getConverter(AudioFormat format) {
910         AudioFloatConverter conv = null;
911         if (format.getFrameSize() == 0)
912             return null;
913         if (format.getFrameSize() !=
914                 ((format.getSampleSizeInBits() + 7) / 8) * format.getChannels()) {
915             return null;
916         }
917         if (format.getEncoding().equals(Encoding.PCM_SIGNED)) {
918             if (format.isBigEndian()) {
919                 if (format.getSampleSizeInBits() <= 8) {
920                     conv = new AudioFloatConversion8S();
921                 } else if (format.getSampleSizeInBits() > 8 &&
922                         format.getSampleSizeInBits() <= 16) {
923                     conv = new AudioFloatConversion16SB();
924                 } else if (format.getSampleSizeInBits() > 16 &&
925                         format.getSampleSizeInBits() <= 24) {
926                     conv = new AudioFloatConversion24SB();
927                 } else if (format.getSampleSizeInBits() > 24 &&
928                         format.getSampleSizeInBits() <= 32) {
```

```
                conv = new AudioFloatConversion32SB();
            } else if (format.getSampleSizeInBits() > 32) {
                conv = new AudioFloatConversion32xSB(((format
                        .getSampleSizeInBits() + 7) / 8) - 4);
            }
        } else {
            if (format.getSampleSizeInBits() <= 8) {
                conv = new AudioFloatConversion8S();
            } else if (format.getSampleSizeInBits() > 8 &&
                    format.getSampleSizeInBits() <= 16) {
                conv = new AudioFloatConversion16SL();
            } else if (format.getSampleSizeInBits() > 16 &&
                    format.getSampleSizeInBits() <= 24) {
                conv = new AudioFloatConversion24SL();
            } else if (format.getSampleSizeInBits() > 24 &&
                    format.getSampleSizeInBits() <= 32) {
                conv = new AudioFloatConversion32SL();
            } else if (format.getSampleSizeInBits() > 32) {
                conv = new AudioFloatConversion32xSL(((format
                        .getSampleSizeInBits() + 7) / 8) - 4);
            }
        }
    } else if (format.getEncoding().equals(Encoding.PCM_UNSIGNED)) {
        if (format.isBigEndian()) {
            if (format.getSampleSizeInBits() <= 8) {
                conv = new AudioFloatConversion8U();
            } else if (format.getSampleSizeInBits() > 8 &&
                    format.getSampleSizeInBits() <= 16) {
                conv = new AudioFloatConversion16UB();
            } else if (format.getSampleSizeInBits() > 16 &&
                    format.getSampleSizeInBits() <= 24) {
                conv = new AudioFloatConversion24UB();
            } else if (format.getSampleSizeInBits() > 24 &&
                    format.getSampleSizeInBits() <= 32) {
                conv = new AudioFloatConversion32UB();
            } else if (format.getSampleSizeInBits() > 32) {
                conv = new AudioFloatConversion32xUB(((
                        format.getSampleSizeInBits() + 7) / 8) - 4);
            }
        } else {
            if (format.getSampleSizeInBits() <= 8) {
                conv = new AudioFloatConversion8U();
            } else if (format.getSampleSizeInBits() > 8 &&
                    format.getSampleSizeInBits() <= 16) {
                conv = new AudioFloatConversion16UL();
            } else if (format.getSampleSizeInBits() > 16 &&
                    format.getSampleSizeInBits() <= 24) {
                conv = new AudioFloatConversion24UL();
            } else if (format.getSampleSizeInBits() > 24 &&
                    format.getSampleSizeInBits() <= 32) {
                conv = new AudioFloatConversion32UL();
            } else if (format.getSampleSizeInBits() > 32) {
                conv = new AudioFloatConversion32xUL(((
                        format.getSampleSizeInBits() + 7) / 8) - 4);
            }
        }
    } else if (format.getEncoding().equals(PCM_FLOAT)) {
        if (format.getSampleSizeInBits() == 32) {
            if (format.isBigEndian())
                conv = new AudioFloatConversion32B();
            else
                conv = new AudioFloatConversion32L();
```

```
            } else if (format.getSampleSizeInBits() == 64) {
                if (format.isBigEndian())
                    conv = new AudioFloatConversion64B();
                else
                    conv = new AudioFloatConversion64L();
            }

        }

        if ((format.getEncoding().equals(Encoding.PCM_SIGNED) ||
                format.getEncoding().equals(Encoding.PCM_UNSIGNED)) &&
                (format.getSampleSizeInBits() % 8 != 0)) {
            conv = new AudioFloatLSBFilter(conv, format);
        }

        if (conv != null)
            conv.format = format;
        return conv;
    }

    private AudioFormat format;

    public AudioFormat getFormat() {
        return format;
    }

    public abstract float[] toFloatArray(byte[] in_buff, int in_offset,
            float[] out_buff, int out_offset, int out_len);

    public float[] toFloatArray(byte[] in_buff, float[] out_buff,
            int out_offset, int out_len) {
        return toFloatArray(in_buff, 0, out_buff, out_offset, out_len);
    }

    public float[] toFloatArray(byte[] in_buff, int in_offset,
            float[] out_buff, int out_len) {
        return toFloatArray(in_buff, in_offset, out_buff, 0, out_len);
    }

    public float[] toFloatArray(byte[] in_buff, float[] out_buff, int out_len) {
        return toFloatArray(in_buff, 0, out_buff, 0, out_len);
    }

    public float[] toFloatArray(byte[] in_buff, float[] out_buff) {
        return toFloatArray(in_buff, 0, out_buff, 0, out_buff.length);
    }

    public abstract byte[] toByteArray(float[] in_buff, int in_offset,
            int in_len, byte[] out_buff, int out_offset);

    public byte[] toByteArray(float[] in_buff, int in_len, byte[] out_buff,
            int out_offset) {
        return toByteArray(in_buff, 0, in_len, out_buff, out_offset);
    }

    public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
            byte[] out_buff) {
        return toByteArray(in_buff, in_offset, in_len, out_buff, 0);
    }

    public byte[] toByteArray(float[] in_buff, int in_len, byte[] out_buff) {
        return toByteArray(in_buff, 0, in_len, out_buff, 0);
```

```
1053        }
1054
1055        public byte[] toByteArray(float[] in_buff, byte[] out_buff) {
1056            return toByteArray(in_buff, 0, in_buff.length, out_buff, 0);
1057        }
1058    }
```

18 com/sun/media/sound/AudioFloatFormatConverter.java

```
1  /*
2   * Copyright 2008 Sun Microsystems, Inc.  All Rights Reserved.
3   * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
4   *
5   * This code is free software; you can redistribute it and/or modify it
6   * under the terms of the GNU General Public License version 2 only, as
7   * published by the Free Software Foundation.  Sun designates this
8   * particular file as subject to the "Classpath" exception as provided
9   * by Sun in the LICENSE file that accompanied this code.
10  *
11  * This code is distributed in the hope that it will be useful, but WITHOUT
12  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
13  * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
14  * version 2 for more details (a copy is included in the LICENSE file that
15  * accompanied this code).
16  *
17  * You should have received a copy of the GNU General Public License version
18  * 2 along with this work; if not, write to the Free Software Foundation,
19  * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
20  *
21  * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
22  * CA 95054 USA or visit www.sun.com if you need additional information or
23  * have any questions.
24  */
25 package com.sun.media.sound;
26
27 import java.io.IOException;
28 import java.io.InputStream;
29 import java.util.ArrayList;
30 import java.util.Arrays;
31
32 import javax.sound.sampled.AudioFormat;
33 import javax.sound.sampled.AudioInputStream;
34 import javax.sound.sampled.AudioSystem;
35 import javax.sound.sampled.AudioFormat.Encoding;
36 import javax.sound.sampled.spi.FormatConversionProvider;
37
38 /**
39  * This class is used to convert between 8,16,24,32 bit signed/unsigned
40  * big/litle endian fixed/floating stereo/mono/multi-channel audio streams and
41  * perform sample-rate conversion if needed.
42  *
43  * @author Karl Helgason
44  */
45 public class AudioFloatFormatConverter extends FormatConversionProvider {
46
47     private static class AudioFloatFormatConverterInputStream extends
48             InputStream {
49         private AudioFloatConverter converter;
50
51         private AudioFloatInputStream stream;
52
53         private float[] readfloatbuffer;
54
55         private int fsize = 0;
56
57         public AudioFloatFormatConverterInputStream(AudioFormat targetFormat,
58                 AudioFloatInputStream stream) {
59             this.stream = stream;
60             converter = AudioFloatConverter.getConverter(targetFormat);
```

```
            fsize = ((targetFormat.getSampleSizeInBits() + 7) / 8);
        }

        public int read() throws IOException {
            byte[] b = new byte[1];
            int ret = read(b);
            if (ret < 0)
                return ret;
            return b[0] & 0xFF;
        }

        public int read(byte[] b, int off, int len) throws IOException {

            int flen = len / fsize;
            if (readfloatbuffer == null || readfloatbuffer.length < flen)
                readfloatbuffer = new float[flen];
            int ret = stream.read(readfloatbuffer, 0, flen);
            if (ret < 0)
                return ret;
            converter.toByteArray(readfloatbuffer, 0, ret, b, off);
            return ret * fsize;
        }

        public int available() throws IOException {
            int ret = stream.available();
            if (ret < 0)
                return ret;
            return ret * fsize;
        }

        public void close() throws IOException {
            stream.close();
        }

        public synchronized void mark(int readlimit) {
            stream.mark(readlimit * fsize);
        }

        public boolean markSupported() {
            return stream.markSupported();
        }

        public synchronized void reset() throws IOException {
            stream.reset();
        }

        public long skip(long n) throws IOException {
            long ret = stream.skip(n / fsize);
            if (ret < 0)
                return ret;
            return ret * fsize;
        }

    }

    private static class AudioFloatInputStreamChannelMixer extends
            AudioFloatInputStream {

        private int targetChannels;

        private int sourceChannels;

```

```
123         private AudioFloatInputStream ais;
124 
125         private AudioFormat targetFormat;
126 
127         private float[] conversion_buffer;
128 
129         public AudioFloatInputStreamChannelMixer(AudioFloatInputStream ais,
130                 int targetChannels) {
131             this.sourceChannels = ais.getFormat().getChannels();
132             this.targetChannels = targetChannels;
133             this.ais = ais;
134             AudioFormat format = ais.getFormat();
135             targetFormat = new AudioFormat(format.getEncoding(), format
136                     .getSampleRate(), format.getSampleSizeInBits(),
137                     targetChannels, (format.getFrameSize() / sourceChannels)
138                             * targetChannels, format.getFrameRate(), format
139                             .isBigEndian());
140         }
141 
142         public int available() throws IOException {
143             return (ais.available() / sourceChannels) * targetChannels;
144         }
145 
146         public void close() throws IOException {
147             ais.close();
148         }
149 
150         public AudioFormat getFormat() {
151             return targetFormat;
152         }
153 
154         public long getFrameLength() {
155             return ais.getFrameLength();
156         }
157 
158         public void mark(int readlimit) {
159             ais.mark((readlimit / targetChannels) * sourceChannels);
160         }
161 
162         public boolean markSupported() {
163             return ais.markSupported();
164         }
165 
166         public int read(float[] b, int off, int len) throws IOException {
167             int len2 = (len / targetChannels) * sourceChannels;
168             if (conversion_buffer == null || conversion_buffer.length < len2)
169                 conversion_buffer = new float[len2];
170             int ret = ais.read(conversion_buffer, 0, len2);
171             if (ret < 0)
172                 return ret;
173             if (sourceChannels == 1) {
174                 int cs = targetChannels;
175                 for (int c = 0; c < targetChannels; c++) {
176                     for (int i = 0, ix = off + c; i < len2; i++, ix += cs) {
177                         b[ix] = conversion_buffer[i];
178                     }
179                 }
180             } else if (targetChannels == 1) {
181                 int cs = sourceChannels;
182                 for (int i = 0, ix = off; i < len2; i += cs, ix++) {
183                     b[ix] = conversion_buffer[i];
184                 }
```

```
                for (int c = 1; c < sourceChannels; c++) {
                    for (int i = c, ix = off; i < len2; i += cs, ix++) {
                        b[ix] += conversion_buffer[i];
                    }
                }
                float vol = 1f / ((float) sourceChannels);
                for (int i = 0, ix = off; i < len2; i += cs, ix++) {
                    b[ix] *= vol;
                }
            } else {
                int minChannels = Math.min(sourceChannels, targetChannels);
                int off_len = off + len;
                int ct = targetChannels;
                int cs = sourceChannels;
                for (int c = 0; c < minChannels; c++) {
                    for (int i = off + c, ix = c; i < off_len; i += ct, ix += cs) {
                        b[i] = conversion_buffer[ix];
                    }
                }
                for (int c = minChannels; c < targetChannels; c++) {
                    for (int i = off + c; i < off_len; i += ct) {
                        b[i] = 0;
                    }
                }
            }
            return (ret / sourceChannels) * targetChannels;
        }

        public void reset() throws IOException {
            ais.reset();
        }

        public long skip(long len) throws IOException {
            long ret = ais.skip((len / targetChannels) * sourceChannels);
            if (ret < 0)
                return ret;
            return (ret / sourceChannels) * targetChannels;
        }

    }

    private static class AudioFloatInputStreamResampler extends
            AudioFloatInputStream {

        private AudioFloatInputStream ais;

        private AudioFormat targetFormat;

        private float[] skipbuffer;

        private SoftAbstractResampler resampler;

        private float[] pitch = new float[1];

        private float[] ibuffer2;

        private float[][] ibuffer;

        private float ibuffer_index = 0;

        private int ibuffer_len = 0;

```

```
247         private int nrofchannels = 0;
248 
249         private float[][] cbuffer;
250 
251         private int buffer_len = 512;
252 
253         private int pad;
254 
255         private int pad2;
256 
257         private float[] ix = new float[1];
258 
259         private int[] ox = new int[1];
260 
261         private float[][] mark_ibuffer = null;
262 
263         private float mark_ibuffer_index = 0;
264 
265         private int mark_ibuffer_len = 0;
266 
267         public AudioFloatInputStreamResampler(AudioFloatInputStream ais,
268                 AudioFormat format) {
269             this.ais = ais;
270             AudioFormat sourceFormat = ais.getFormat();
271             targetFormat = new AudioFormat(sourceFormat.getEncoding(), format
272                     .getSampleRate(), sourceFormat.getSampleSizeInBits(),
273                     sourceFormat.getChannels(), sourceFormat.getFrameSize(),
274                     format.getSampleRate(), sourceFormat.isBigEndian());
275             nrofchannels = targetFormat.getChannels();
276             Object interpolation = format.getProperty("interpolation");
277             if (interpolation != null && (interpolation instanceof String)) {
278                 String resamplerType = (String) interpolation;
279                 if (resamplerType.equalsIgnoreCase("point"))
280                     this.resampler = new SoftPointResampler();
281                 if (resamplerType.equalsIgnoreCase("linear"))
282                     this.resampler = new SoftLinearResampler2();
283                 if (resamplerType.equalsIgnoreCase("linear1"))
284                     this.resampler = new SoftLinearResampler();
285                 if (resamplerType.equalsIgnoreCase("linear2"))
286                     this.resampler = new SoftLinearResampler2();
287                 if (resamplerType.equalsIgnoreCase("cubic"))
288                     this.resampler = new SoftCubicResampler();
289                 if (resamplerType.equalsIgnoreCase("lanczos"))
290                     this.resampler = new SoftLanczosResampler();
291                 if (resamplerType.equalsIgnoreCase("sinc"))
292                     this.resampler = new SoftSincResampler();
293             }
294             if (resampler == null)
295                 resampler = new SoftLinearResampler2(); // new
296                                                         // SoftLinearResampler2();
297             pitch[0] = sourceFormat.getSampleRate() / format.getSampleRate();
298             pad = resampler.getPadding();
299             pad2 = pad * 2;
300             ibuffer = new float[nrofchannels][buffer_len + pad2];
301             ibuffer2 = new float[nrofchannels * buffer_len];
302             ibuffer_index = buffer_len + pad;
303             ibuffer_len = buffer_len;
304         }
305 
306         public int available() throws IOException {
307             return 0;
308         }
```

```
309
310         public void close() throws IOException {
311             ais.close();
312         }
313
314         public AudioFormat getFormat() {
315             return targetFormat;
316         }
317
318         public long getFrameLength() {
319             return AudioSystem.NOT_SPECIFIED; // ais.getFrameLength();
320         }
321
322         public void mark(int readlimit) {
323             ais.mark((int) (readlimit * pitch[0]));
324             mark_ibuffer_index = ibuffer_index;
325             mark_ibuffer_len = ibuffer_len;
326             if (mark_ibuffer == null) {
327                 mark_ibuffer = new float[ibuffer.length][ibuffer[0].length];
328             }
329             for (int c = 0; c < ibuffer.length; c++) {
330                 float[] from = ibuffer[c];
331                 float[] to = mark_ibuffer[c];
332                 for (int i = 0; i < to.length; i++) {
333                     to[i] = from[i];
334                 }
335             }
336         }
337
338         public boolean markSupported() {
339             return ais.markSupported();
340         }
341
342         private void readNextBuffer() throws IOException {
343
344             if (ibuffer_len == -1)
345                 return;
346
347             for (int c = 0; c < nrofchannels; c++) {
348                 float[] buff = ibuffer[c];
349                 int buffer_len_pad = ibuffer_len + pad2;
350                 for (int i = ibuffer_len, ix = 0; i < buffer_len_pad; i++, ix++) {
351                     buff[ix] = buff[i];
352                 }
353             }
354
355             ibuffer_index -= (ibuffer_len);
356
357             ibuffer_len = ais.read(ibuffer2);
358             if (ibuffer_len >= 0) {
359                 while (ibuffer_len < ibuffer2.length) {
360                     int ret = ais.read(ibuffer2, ibuffer_len, ibuffer2.length
361                             - ibuffer_len);
362                     if (ret == -1)
363                         break;
364                     ibuffer_len += ret;
365                 }
366                 Arrays.fill(ibuffer2, ibuffer_len, ibuffer2.length, 0);
367                 ibuffer_len /= nrofchannels;
368             } else {
369                 Arrays.fill(ibuffer2, 0, ibuffer2.length, 0);
370             }
```

```
371
372             int ibuffer2_len = ibuffer2.length;
373             for (int c = 0; c < nrofchannels; c++) {
374                 float[] buff = ibuffer[c];
375                 for (int i = c, ix = pad2; i < ibuffer2_len; i += nrofchannels, ix++) {
376                     buff[ix] = ibuffer2[i];
377                 }
378             }
379
380         }
381
382         public int read(float[] b, int off, int len) throws IOException {
383
384             if (cbuffer == null || cbuffer[0].length < len / nrofchannels) {
385                 cbuffer = new float[nrofchannels][len / nrofchannels];
386             }
387             if (ibuffer_len == -1)
388                 return -1;
389             if (len < 0)
390                 return 0;
391             int offlen = off + len;
392             int remain = len / nrofchannels;
393             int destPos = 0;
394             int in_end = ibuffer_len;
395             while (remain > 0) {
396                 if (ibuffer_len >= 0) {
397                     if (ibuffer_index >= (ibuffer_len + pad))
398                         readNextBuffer();
399                     in_end = ibuffer_len + pad;
400                 }
401
402                 if (ibuffer_len < 0) {
403                     in_end = pad2;
404                     if (ibuffer_index >= in_end)
405                         break;
406                 }
407
408                 if (ibuffer_index < 0)
409                     break;
410                 int preDestPos = destPos;
411                 for (int c = 0; c < nrofchannels; c++) {
412                     ix[0] = ibuffer_index;
413                     ox[0] = destPos;
414                     float[] buff = ibuffer[c];
415                     resampler.interpolate(buff, ix, in_end, pitch, 0,
416                             cbuffer[c], ox, len / nrofchannels);
417                 }
418                 ibuffer_index = ix[0];
419                 destPos = ox[0];
420                 remain -= destPos - preDestPos;
421             }
422             for (int c = 0; c < nrofchannels; c++) {
423                 int ix = 0;
424                 float[] buff = cbuffer[c];
425                 for (int i = c + off; i < offlen; i += nrofchannels) {
426                     b[i] = buff[ix++];
427                 }
428             }
429             return len - remain * nrofchannels;
430         }
431
432         public void reset() throws IOException {
```

```
            ais.reset();
            if (mark_ibuffer == null)
                return;
            ibuffer_index = mark_ibuffer_index;
            ibuffer_len = mark_ibuffer_len;
            for (int c = 0; c < ibuffer.length; c++) {
                float[] from = mark_ibuffer[c];
                float[] to = ibuffer[c];
                for (int i = 0; i < to.length; i++) {
                    to[i] = from[i];
                }
            }

        }

        public long skip(long len) throws IOException {
            if (len < 0)
                return 0;
            if (skipbuffer == null)
                skipbuffer = new float[1024 * targetFormat.getFrameSize()];
            float[] l_skipbuffer = skipbuffer;
            long remain = len;
            while (remain > 0) {
                int ret = read(l_skipbuffer, 0, (int) Math.min(remain,
                        skipbuffer.length));
                if (ret < 0) {
                    if (remain == len)
                        return ret;
                    break;
                }
                remain -= ret;
            }
            return len - remain;

        }

    }

    private Encoding[] formats = { Encoding.PCM_SIGNED, Encoding.PCM_UNSIGNED,
            AudioFloatConverter.PCM_FLOAT };

    public AudioInputStream getAudioInputStream(Encoding targetEncoding,
            AudioInputStream sourceStream) {
        if (sourceStream.getFormat().getEncoding().equals(targetEncoding))
            return sourceStream;
        AudioFormat format = sourceStream.getFormat();
        int channels = format.getChannels();
        Encoding encoding = targetEncoding;
        float samplerate = format.getSampleRate();
        int bits = format.getSampleSizeInBits();
        boolean bigendian = format.isBigEndian();
        if (targetEncoding.equals(AudioFloatConverter.PCM_FLOAT))
            bits = 32;
        AudioFormat targetFormat = new AudioFormat(encoding, samplerate, bits,
                channels, channels * bits / 8, samplerate, bigendian);
        return getAudioInputStream(targetFormat, sourceStream);
    }

    public AudioInputStream getAudioInputStream(AudioFormat targetFormat,
            AudioInputStream sourceStream) {
        if (!isConversionSupported(targetFormat, sourceStream.getFormat()))
            throw new IllegalArgumentException("Unsupported conversion: "
```

```
                + sourceStream.getFormat().toString() + " to "
                + targetFormat.toString());
        return getAudioInputStream(targetFormat, AudioFloatInputStream
                .getInputStream(sourceStream));
    }

    public AudioInputStream getAudioInputStream(AudioFormat targetFormat,
            AudioFloatInputStream sourceStream) {

        if (!isConversionSupported(targetFormat, sourceStream.getFormat()))
            throw new IllegalArgumentException("Unsupported conversion: "
                    + sourceStream.getFormat().toString() + " to "
                    + targetFormat.toString());
        if (targetFormat.getChannels() != sourceStream.getFormat()
                .getChannels())
            sourceStream = new AudioFloatInputStreamChannelMixer(sourceStream,
                    targetFormat.getChannels());
        if (Math.abs(targetFormat.getSampleRate()
                - sourceStream.getFormat().getSampleRate()) > 0.000001)
            sourceStream = new AudioFloatInputStreamResampler(sourceStream,
                    targetFormat);
        return new AudioInputStream(new AudioFloatFormatConverterInputStream(
                targetFormat, sourceStream), targetFormat, sourceStream
                .getFrameLength());
    }

    public Encoding[] getSourceEncodings() {
        return new Encoding[] { Encoding.PCM_SIGNED, Encoding.PCM_UNSIGNED,
                AudioFloatConverter.PCM_FLOAT };
    }

    public Encoding[] getTargetEncodings() {
        return new Encoding[] { Encoding.PCM_SIGNED, Encoding.PCM_UNSIGNED,
                AudioFloatConverter.PCM_FLOAT };
    }

    public Encoding[] getTargetEncodings(AudioFormat sourceFormat) {
        if (AudioFloatConverter.getConverter(sourceFormat) == null)
            return new Encoding[0];
        return new Encoding[] { Encoding.PCM_SIGNED, Encoding.PCM_UNSIGNED,
                AudioFloatConverter.PCM_FLOAT };
    }

    public AudioFormat[] getTargetFormats(Encoding targetEncoding,
            AudioFormat sourceFormat) {
        if (AudioFloatConverter.getConverter(sourceFormat) == null)
            return new AudioFormat[0];
        int channels = sourceFormat.getChannels();

        ArrayList<AudioFormat> formats = new ArrayList<AudioFormat>();

        if (targetEncoding.equals(Encoding.PCM_SIGNED))
            formats.add(new AudioFormat(Encoding.PCM_SIGNED,
                    AudioSystem.NOT_SPECIFIED, 8, channels, channels,
                    AudioSystem.NOT_SPECIFIED, false));
        if (targetEncoding.equals(Encoding.PCM_UNSIGNED))
            formats.add(new AudioFormat(Encoding.PCM_UNSIGNED,
                    AudioSystem.NOT_SPECIFIED, 8, channels, channels,
                    AudioSystem.NOT_SPECIFIED, false));

        for (int bits = 16; bits < 32; bits += 8) {
            if (targetEncoding.equals(Encoding.PCM_SIGNED)) {
```

```
                formats.add(new AudioFormat(Encoding.PCM_SIGNED,
                        AudioSystem.NOT_SPECIFIED, bits, channels, channels
                                * bits / 8, AudioSystem.NOT_SPECIFIED, false));
                formats.add(new AudioFormat(Encoding.PCM_SIGNED,
                        AudioSystem.NOT_SPECIFIED, bits, channels, channels
                                * bits / 8, AudioSystem.NOT_SPECIFIED, true));
            }
            if (targetEncoding.equals(Encoding.PCM_UNSIGNED)) {
                formats.add(new AudioFormat(Encoding.PCM_UNSIGNED,
                        AudioSystem.NOT_SPECIFIED, bits, channels, channels
                                * bits / 8, AudioSystem.NOT_SPECIFIED, true));
                formats.add(new AudioFormat(Encoding.PCM_UNSIGNED,
                        AudioSystem.NOT_SPECIFIED, bits, channels, channels
                                * bits / 8, AudioSystem.NOT_SPECIFIED, false));
            }
        }

        if (targetEncoding.equals(AudioFloatConverter.PCM_FLOAT)) {
            formats.add(new AudioFormat(AudioFloatConverter.PCM_FLOAT,
                    AudioSystem.NOT_SPECIFIED, 32, channels, channels * 4,
                    AudioSystem.NOT_SPECIFIED, false));
            formats.add(new AudioFormat(AudioFloatConverter.PCM_FLOAT,
                    AudioSystem.NOT_SPECIFIED, 32, channels, channels * 4,
                    AudioSystem.NOT_SPECIFIED, true));
            formats.add(new AudioFormat(AudioFloatConverter.PCM_FLOAT,
                    AudioSystem.NOT_SPECIFIED, 64, channels, channels * 8,
                    AudioSystem.NOT_SPECIFIED, false));
            formats.add(new AudioFormat(AudioFloatConverter.PCM_FLOAT,
                    AudioSystem.NOT_SPECIFIED, 64, channels, channels * 8,
                    AudioSystem.NOT_SPECIFIED, true));
        }

        return formats.toArray(new AudioFormat[formats.size()]);
    }

    public boolean isConversionSupported(AudioFormat targetFormat,
            AudioFormat sourceFormat) {
        if (AudioFloatConverter.getConverter(sourceFormat) == null)
            return false;
        if (AudioFloatConverter.getConverter(targetFormat) == null)
            return false;
        if (sourceFormat.getChannels() <= 0)
            return false;
        if (targetFormat.getChannels() <= 0)
            return false;
        return true;
    }

    public boolean isConversionSupported(Encoding targetEncoding,
            AudioFormat sourceFormat) {
        if (AudioFloatConverter.getConverter(sourceFormat) == null)
            return false;
        for (int i = 0; i < formats.length; i++) {
            if (targetEncoding.equals(formats[i]))
                return true;
        }
        return false;
    }

}
```

19 com/sun/media/sound/AudioFloatInputStream.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

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

import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.UnsupportedAudioFileException;

/**
 * This class is used to create AudioFloatInputStream from AudioInputStream and
 * byte buffers.
 *
 * @author Karl Helgason
 */
public abstract class AudioFloatInputStream {

    private static class BytaArrayAudioFloatInputStream
            extends AudioFloatInputStream {

        private int pos = 0;
        private int markpos = 0;
        private AudioFloatConverter converter;
        private AudioFormat format;
        private byte[] buffer;
        private int buffer_offset;
        private int buffer_len;
        private int framesize_pc;

        public BytaArrayAudioFloatInputStream(AudioFloatConverter converter,
                byte[] buffer, int offset, int len) {
            this.converter = converter;
```

```
61              this.format = converter.getFormat();
62              this.buffer = buffer;
63              this.buffer_offset = offset;
64              framesize_pc = format.getFrameSize() / format.getChannels();
65              this.buffer_len = len / framesize_pc;
66
67          }
68
69          public AudioFormat getFormat() {
70              return format;
71          }
72
73          public long getFrameLength() {
74              return buffer_len;// / format.getFrameSize();
75          }
76
77          public int read(float[] b, int off, int len) throws IOException {
78              if (b == null)
79                  throw new NullPointerException();
80              if (off < 0 || len < 0 || len > b.length - off)
81                  throw new IndexOutOfBoundsException();
82              if (pos >= buffer_len)
83                  return -1;
84              if (len == 0)
85                  return 0;
86              if (pos + len > buffer_len)
87                  len = buffer_len - pos;
88              converter.toFloatArray(buffer, buffer_offset + pos * framesize_pc,
89                      b, off, len);
90              pos += len;
91              return len;
92          }
93
94          public long skip(long len) throws IOException {
95              if (pos >= buffer_len)
96                  return -1;
97              if (len <= 0)
98                  return 0;
99              if (pos + len > buffer_len)
100                 len = buffer_len - pos;
101             pos += len;
102             return len;
103         }
104
105         public int available() throws IOException {
106             return buffer_len - pos;
107         }
108
109         public void close() throws IOException {
110         }
111
112         public void mark(int readlimit) {
113             markpos = pos;
114         }
115
116         public boolean markSupported() {
117             return true;
118         }
119
120         public void reset() throws IOException {
121             pos = markpos;
122         }
```

```
123     }
124
125     private static class DirectAudioFloatInputStream
126             extends AudioFloatInputStream {
127
128         private AudioInputStream stream;
129         private AudioFloatConverter converter;
130         private int framesize_pc; // framesize / channels
131         private byte[] buffer;
132
133         public DirectAudioFloatInputStream(AudioInputStream stream) {
134             converter = AudioFloatConverter.getConverter(stream.getFormat());
135             if (converter == null) {
136                 AudioFormat format = stream.getFormat();
137                 AudioFormat newformat;
138
139                 AudioFormat[] formats = AudioSystem.getTargetFormats(
140                         AudioFormat.Encoding.PCM_SIGNED, format);
141                 if (formats.length != 0) {
142                     newformat = formats[0];
143                 } else {
144                     float samplerate = format.getSampleRate();
145                     int samplesizeinbits = format.getSampleSizeInBits();
146                     int framesize = format.getFrameSize();
147                     float framerate = format.getFrameRate();
148                     samplesizeinbits = 16;
149                     framesize = format.getChannels() * (samplesizeinbits / 8);
150                     framerate = samplerate;
151
152                     newformat = new AudioFormat(
153                             AudioFormat.Encoding.PCM_SIGNED, samplerate,
154                             samplesizeinbits, format.getChannels(), framesize,
155                             framerate, false);
156                 }
157
158                 stream = AudioSystem.getAudioInputStream(newformat, stream);
159                 converter = AudioFloatConverter.getConverter(stream.getFormat());
160             }
161             framesize_pc = stream.getFormat().getFrameSize()
162                     / stream.getFormat().getChannels();
163             this.stream = stream;
164         }
165
166         public AudioFormat getFormat() {
167             return stream.getFormat();
168         }
169
170         public long getFrameLength() {
171             return stream.getFrameLength();
172         }
173
174         public int read(float[] b, int off, int len) throws IOException {
175             int b_len = len * framesize_pc;
176             if (buffer == null || buffer.length < b_len)
177                 buffer = new byte[b_len];
178             int ret = stream.read(buffer, 0, b_len);
179             if (ret == -1)
180                 return -1;
181             converter.toFloatArray(buffer, b, off, ret / framesize_pc);
182             return ret / framesize_pc;
183         }
184
```

```
185         public long skip(long len) throws IOException {
186             long b_len = len * framesize_pc;
187             long ret = stream.skip(b_len);
188             if (ret == -1)
189                 return -1;
190             return ret / framesize_pc;
191         }
192 
193         public int available() throws IOException {
194             return stream.available() / framesize_pc;
195         }
196 
197         public void close() throws IOException {
198             stream.close();
199         }
200 
201         public void mark(int readlimit) {
202             stream.mark(readlimit * framesize_pc);
203         }
204 
205         public boolean markSupported() {
206             return stream.markSupported();
207         }
208 
209         public void reset() throws IOException {
210             stream.reset();
211         }
212     }
213 
214     public static AudioFloatInputStream getInputStream(URL url)
215             throws UnsupportedAudioFileException, IOException {
216         return new DirectAudioFloatInputStream(AudioSystem
217                 .getAudioInputStream(url));
218     }
219 
220     public static AudioFloatInputStream getInputStream(File file)
221             throws UnsupportedAudioFileException, IOException {
222         return new DirectAudioFloatInputStream(AudioSystem
223                 .getAudioInputStream(file));
224     }
225 
226     public static AudioFloatInputStream getInputStream(InputStream stream)
227             throws UnsupportedAudioFileException, IOException {
228         return new DirectAudioFloatInputStream(AudioSystem
229                 .getAudioInputStream(stream));
230     }
231 
232     public static AudioFloatInputStream getInputStream(
233             AudioInputStream stream) {
234         return new DirectAudioFloatInputStream(stream);
235     }
236 
237     public static AudioFloatInputStream getInputStream(AudioFormat format,
238             byte[] buffer, int offset, int len) {
239         AudioFloatConverter converter = AudioFloatConverter
240                 .getConverter(format);
241         if (converter != null)
242             return new BytaArrayAudioFloatInputStream(converter, buffer,
243                     offset, len);
244 
245         InputStream stream = new ByteArrayInputStream(buffer, offset, len);
246         long aLen = format.getFrameSize() == AudioSystem.NOT_SPECIFIED
```

```
                ? AudioSystem.NOT_SPECIFIED : len / format.getFrameSize();
        AudioInputStream astream = new AudioInputStream(stream, format, aLen);
        return getInputStream(astream);
    }

    public abstract AudioFormat getFormat();

    public abstract long getFrameLength();

    public abstract int read(float[] b, int off, int len) throws IOException;

    public int read(float[] b) throws IOException {
        return read(b, 0, b.length);
    }

    public float read() throws IOException {
        float[] b = new float[1];
        int ret = read(b, 0, 1);
        if (ret == -1 || ret == 0)
            return 0;
        return b[0];
    }

    public abstract long skip(long len) throws IOException;

    public abstract int available() throws IOException;

    public abstract void close() throws IOException;

    public abstract void mark(int readlimit);

    public abstract boolean markSupported();

    public abstract void reset() throws IOException;
}
```

20 com/sun/media/sound/AudioSynthesizer.java

```
1  /*
2   * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
3   * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
4   *
5   * This code is free software; you can redistribute it and/or modify it
6   * under the terms of the GNU General Public License version 2 only, as
7   * published by the Free Software Foundation.  Sun designates this
8   * particular file as subject to the "Classpath" exception as provided
9   * by Sun in the LICENSE file that accompanied this code.
10  *
11  * This code is distributed in the hope that it will be useful, but WITHOUT
12  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
13  * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
14  * version 2 for more details (a copy is included in the LICENSE file that
15  * accompanied this code).
16  *
17  * You should have received a copy of the GNU General Public License version
18  * 2 along with this work; if not, write to the Free Software Foundation,
19  * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
20  *
21  * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
22  * CA 95054 USA or visit www.sun.com if you need additional information or
23  * have any questions.
24  */
25 package com.sun.media.sound;
26
27 import java.util.Map;
28 import javax.sound.midi.MidiUnavailableException;
29 import javax.sound.midi.Synthesizer;
30 import javax.sound.sampled.AudioFormat;
31 import javax.sound.sampled.AudioInputStream;
32 import javax.sound.sampled.SourceDataLine;
33
34 /**
35  * <code>AudioSynthesizer</code> is a <code>Synthesizer</code>
36  * which renders it's output audio into <code>SourceDataLine</code>
37  * or <code>AudioInputStream</code>.
38  *
39  * @see MidiSystem#getSynthesizer
40  * @see Synthesizer
41  *
42  * @author Karl Helgason
43  */
44 public interface AudioSynthesizer extends Synthesizer {
45
46     /**
47      * Obtains the current format (encoding, sample rate, number of channels,
48      * etc.) of the synthesizer audio data.
49      *
50      * <p>If the synthesizer is not open and has never been opened, it returns
51      * the default format.
52      *
53      * @return current audio data format
54      * @see AudioFormat
55      */
56     public AudioFormat getFormat();
57
58     /**
59      * Gets information about the possible properties for the synthesizer.
60      *
```

```
     * @param info a proposed list of tag/value pairs that will be sent on open.
     * @return an array of <code>AudioSynthesizerPropertyInfo</code> objects
     * describing possible properties. This array may be an empty array if
     * no properties are required.
     */
    public AudioSynthesizerPropertyInfo[] getPropertyInfo(
            Map<String, Object> info);

    /**
     * Opens the synthesizer and starts rendering audio into
     * <code>SourceDataLine</code>.
     *
     * <p>An application opening a synthesizer explicitly with this call
     * has to close the synthesizer by calling {@link #close}. This is
     * necessary to release system resources and allow applications to
     * exit cleanly.
     *
     * <p>Note that some synthesizers, once closed, cannot be reopened.
     * Attempts to reopen such a synthesizer will always result in
     * a <code>MidiUnavailableException</code>.
     *
     * @param line which <code>AudioSynthesizer</code> writes output audio into.
     * If <code>line</code> is null, then line from system default mixer is used.
     * @param info a <code>Map<String,Object></code> object containing
     * properties for additional configuration supported by synthesizer.
     * If <code>info</code> is null then default settings are used.
     *
     * @throws MidiUnavailableException thrown if the synthesizer cannot be
     * opened due to resource restrictions.
     * @throws SecurityException thrown if the synthesizer cannot be
     * opened due to security restrictions.
     *
     * @see #close
     * @see #isOpen
     */
    public void open(SourceDataLine line, Map<String, Object> info)
            throws MidiUnavailableException;

    /**
     * Opens the synthesizer and renders audio into returned
     * <code>AudioInputStream</code>.
     *
     * <p>An application opening a synthesizer explicitly with this call
     * has to close the synthesizer by calling {@link #close}. This is
     * necessary to release system resources and allow applications to
     * exit cleanly.
     *
     * <p>Note that some synthesizers, once closed, cannot be reopened.
     * Attempts to reopen such a synthesizer will always result in
     * a <code>MidiUnavailableException<code>.
     *
     * @param targetFormat specifies the <code>AudioFormat</code>
     * used in returned <code>AudioInputStream</code>.
     * @param info a <code>Map<String,Object></code> object containing
     * properties for additional configuration supported by synthesizer.
     * If <code>info</code> is null then default settings are used.
     *
     * @throws MidiUnavailableException thrown if the synthesizer cannot be
     * opened due to resource restrictions.
     * @throws SecurityException thrown if the synthesizer cannot be
     * opened due to security restrictions.
     *
```

```
123         * @see #close
124         * @see #isOpen
125         */
126        public AudioInputStream openStream(AudioFormat targetFormat,
127                Map<String, Object> info) throws MidiUnavailableException;
128    }
```

21 com/sun/media/sound/AudioSynthesizerPropertyInfo.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * Information about property used in  opening <code>AudioSynthesizer</code>.
 *
 * @author Karl Helgason
 */
public class AudioSynthesizerPropertyInfo {

    /**
     * Constructs a <code>AudioSynthesizerPropertyInfo</code> object with a given
     * name and value. The <code>description</code> and <code>choices</code>
     * are intialized by <code>null</code> values.
     *
     * @param name the name of the property
     * @param value the current value or class used for values.
     *
     */
    public AudioSynthesizerPropertyInfo(String name, Object value) {
        this.name = name;
        if (value instanceof Class)
            valueClass = (Class)value;
        else
        {
            this.value = value;
            if (value != null)
                valueClass = value.getClass();
        }
    }
    /**
     * The name of the property.
     */
    public String name;
    /**
     * A brief description of the property, which may be null.
     */
```

```
61      public String description = null;
62      /**
63       * The <code>value</code> field specifies the current value of
64       * the property.
65       */
66      public Object value = null;
67      /**
68       * The <code>valueClass</code> field specifies class
69       * used in <code>value</code> field.
70       */
71      public Class valueClass = null;
72      /**
73       * An array of possible values if the value for the field
74       * <code>AudioSynthesizerPropertyInfo.value</code> may be selected
75       * from a particular set of values; otherwise null.
76       */
77      public Object[] choices = null;
78
79  }
```

22 com/sun/media/sound/DLSInfo.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * This class is used to store information  to describe soundbanks, instruments
 * and samples. It is stored inside a "INFO" List Chunk inside DLS files.
 *
 * @author Karl Helgason
 */
public class DLSInfo {

    /**
     * (INAM) Title or subject.
     */
    public String name = "untitled";
    /**
     * (ICRD) Date of creation, the format is: YYYY-MM-DD.
     *        For example 2007-01-01 for 1. january of year 2007.
     */
    public String creationDate = null;
    /**
     * (IENG) Name of engineer who created the object.
     */
    public String engineers = null;
    /**
     * (IPRD) Name of the product which the object is intended for.
     */
    public String product = null;
    /**
     * (ICOP) Copyright information.
     */
    public String copyright = null;
    /**
     * (ICMT) General comments. Doesn't contain newline characters.
     */
    public String comments = null;
    /**
```

```
     * (ISFT) Name of software package used to create the file.
     */
    public String tools = null;
    /**
     * (IARL) Where content is archived.
     */
    public String archival_location = null;
    /**
     * (IART) Artists of original content.
     */
    public String artist = null;
    /**
     * (ICMS) Names of persons or orginizations who commissioned the file.
     */
    public String commissioned = null;
    /**
     * (IGNR) Genre of the work.
     *        Example: jazz, classical, rock, etc.
     */
    public String genre = null;
    /**
     * (IKEY) List of keyword that describe the content.
     *        Examples: FX, bird, piano, etc.
     */
    public String keywords = null;
    /**
     * (IMED) Describes original medium of the data.
     *        For example: record, CD, etc.
     */
    public String medium = null;
    /**
     * (ISBJ) Description of the content.
     */
    public String subject = null;
    /**
     * (ISRC) Name of person or orginization who supplied
     *        orginal material for the file.
     */
    public String source = null;
    /**
     * (ISRF) Source media for sample data is from.
     *        For example: CD, TV, etc.
     */
    public String source_form = null;
    /**
     * (ITCH) Technician who sample the file/object.
     */
    public String technician = null;
}
```

23 com/sun/media/sound/DLSInstrument.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import javax.sound.midi.Patch;

/**
 * This class is used to store information to describe instrument.
 * It contains list of regions and modulators.
 * It is stored inside a "ins " List Chunk inside DLS files.
 * In the DLS documentation a modulator is called articulator.
 *
 * @author Karl Helgason
 */
public class DLSInstrument extends ModelInstrument {

    protected int preset = 0;
    protected int bank = 0;
    protected boolean druminstrument = false;
    protected byte[] guid = null;
    protected DLSInfo info = new DLSInfo();
    protected List<DLSRegion> regions = new ArrayList<DLSRegion>();
    protected List<DLSModulator> modulators = new ArrayList<DLSModulator>();

    public DLSInstrument() {
        super(null, null, null, null);
    }

    public DLSInstrument(DLSSoundbank soundbank) {
        super(soundbank, null, null, null);
    }

    public DLSInfo getInfo() {
```

```
61          return info;
62      }
63
64      public String getName() {
65          return info.name;
66      }
67
68      public void setName(String name) {
69          info.name = name;
70      }
71
72      public ModelPatch getPatch() {
73          return new ModelPatch(bank, preset, druminstrument);
74      }
75
76      public void setPatch(Patch patch) {
77          if (patch instanceof ModelPatch && ((ModelPatch)patch).isPercussion()) {
78              druminstrument = true;
79              bank = patch.getBank();
80              preset = patch.getProgram();
81          } else {
82              druminstrument = false;
83              bank = patch.getBank();
84              preset = patch.getProgram();
85          }
86      }
87
88      public Object getData() {
89          return null;
90      }
91
92      public List<DLSRegion> getRegions() {
93          return regions;
94      }
95
96      public List<DLSModulator> getModulators() {
97          return modulators;
98      }
99
100     public String toString() {
101         if (druminstrument)
102             return "Drumkit: " + info.name
103                     + " bank #" + bank + " preset #" + preset;
104         else
105             return "Instrument: " + info.name
106                     + " bank #" + bank + " preset #" + preset;
107     }
108
109     private ModelIdentifier convertToModelDest(int dest) {
110         if (dest == DLSModulator.CONN_DST_NONE)
111             return null;
112         if (dest == DLSModulator.CONN_DST_GAIN)
113             return ModelDestination.DESTINATION_GAIN;
114         if (dest == DLSModulator.CONN_DST_PITCH)
115             return ModelDestination.DESTINATION_PITCH;
116         if (dest == DLSModulator.CONN_DST_PAN)
117             return ModelDestination.DESTINATION_PAN;
118
119         if (dest == DLSModulator.CONN_DST_LFO_FREQUENCY)
120             return ModelDestination.DESTINATION_LFO1_FREQ;
121         if (dest == DLSModulator.CONN_DST_LFO_STARTDELAY)
122             return ModelDestination.DESTINATION_LFO1_DELAY;
```

```
123
124         if (dest == DLSModulator.CONN_DST_EG1_ATTACKTIME)
125             return ModelDestination.DESTINATION_EG1_ATTACK;
126         if (dest == DLSModulator.CONN_DST_EG1_DECAYTIME)
127             return ModelDestination.DESTINATION_EG1_DECAY;
128         if (dest == DLSModulator.CONN_DST_EG1_RELEASETIME)
129             return ModelDestination.DESTINATION_EG1_RELEASE;
130         if (dest == DLSModulator.CONN_DST_EG1_SUSTAINLEVEL)
131             return ModelDestination.DESTINATION_EG1_SUSTAIN;
132
133         if (dest == DLSModulator.CONN_DST_EG2_ATTACKTIME)
134             return ModelDestination.DESTINATION_EG2_ATTACK;
135         if (dest == DLSModulator.CONN_DST_EG2_DECAYTIME)
136             return ModelDestination.DESTINATION_EG2_DECAY;
137         if (dest == DLSModulator.CONN_DST_EG2_RELEASETIME)
138             return ModelDestination.DESTINATION_EG2_RELEASE;
139         if (dest == DLSModulator.CONN_DST_EG2_SUSTAINLEVEL)
140             return ModelDestination.DESTINATION_EG2_SUSTAIN;
141
142         // DLS2 Destinations
143         if (dest == DLSModulator.CONN_DST_KEYNUMBER)
144             return ModelDestination.DESTINATION_KEYNUMBER;
145
146         if (dest == DLSModulator.CONN_DST_CHORUS)
147             return ModelDestination.DESTINATION_CHORUS;
148         if (dest == DLSModulator.CONN_DST_REVERB)
149             return ModelDestination.DESTINATION_REVERB;
150
151         if (dest == DLSModulator.CONN_DST_VIB_FREQUENCY)
152             return ModelDestination.DESTINATION_LFO2_FREQ;
153         if (dest == DLSModulator.CONN_DST_VIB_STARTDELAY)
154             return ModelDestination.DESTINATION_LFO2_DELAY;
155
156         if (dest == DLSModulator.CONN_DST_EG1_DELAYTIME)
157             return ModelDestination.DESTINATION_EG1_DELAY;
158         if (dest == DLSModulator.CONN_DST_EG1_HOLDTIME)
159             return ModelDestination.DESTINATION_EG1_HOLD;
160         if (dest == DLSModulator.CONN_DST_EG1_SHUTDOWNTIME)
161             return ModelDestination.DESTINATION_EG1_SHUTDOWN;
162
163         if (dest == DLSModulator.CONN_DST_EG2_DELAYTIME)
164             return ModelDestination.DESTINATION_EG2_DELAY;
165         if (dest == DLSModulator.CONN_DST_EG2_HOLDTIME)
166             return ModelDestination.DESTINATION_EG2_HOLD;
167
168         if (dest == DLSModulator.CONN_DST_FILTER_CUTOFF)
169             return ModelDestination.DESTINATION_FILTER_FREQ;
170         if (dest == DLSModulator.CONN_DST_FILTER_Q)
171             return ModelDestination.DESTINATION_FILTER_Q;
172
173         return null;
174     }
175
176     private ModelIdentifier convertToModelSrc(int src) {
177         if (src == DLSModulator.CONN_SRC_NONE)
178             return null;
179
180         if (src == DLSModulator.CONN_SRC_LFO)
181             return ModelSource.SOURCE_LFO1;
182         if (src == DLSModulator.CONN_SRC_KEYONVELOCITY)
183             return ModelSource.SOURCE_NOTEON_VELOCITY;
184         if (src == DLSModulator.CONN_SRC_KEYNUMBER)
```

```
            return ModelSource.SOURCE_NOTEON_KEYNUMBER;
        if (src == DLSModulator.CONN_SRC_EG1)
            return ModelSource.SOURCE_EG1;
        if (src == DLSModulator.CONN_SRC_EG2)
            return ModelSource.SOURCE_EG2;
        if (src == DLSModulator.CONN_SRC_PITCHWHEEL)
            return ModelSource.SOURCE_MIDI_PITCH;
        if (src == DLSModulator.CONN_SRC_CC1)
            return new ModelIdentifier("midi_cc", "1", 0);
        if (src == DLSModulator.CONN_SRC_CC7)
            return new ModelIdentifier("midi_cc", "7", 0);
        if (src == DLSModulator.CONN_SRC_CC10)
            return new ModelIdentifier("midi_cc", "10", 0);
        if (src == DLSModulator.CONN_SRC_CC11)
            return new ModelIdentifier("midi_cc", "11", 0);
        if (src == DLSModulator.CONN_SRC_RPN0)
            return new ModelIdentifier("midi_rpn", "0", 0);
        if (src == DLSModulator.CONN_SRC_RPN1)
            return new ModelIdentifier("midi_rpn", "1", 0);

        if (src == DLSModulator.CONN_SRC_POLYPRESSURE)
            return ModelSource.SOURCE_MIDI_POLY_PRESSURE;
        if (src == DLSModulator.CONN_SRC_CHANNELPRESSURE)
            return ModelSource.SOURCE_MIDI_CHANNEL_PRESSURE;
        if (src == DLSModulator.CONN_SRC_VIBRATO)
            return ModelSource.SOURCE_LFO2;
        if (src == DLSModulator.CONN_SRC_MONOPRESSURE)
            return ModelSource.SOURCE_MIDI_CHANNEL_PRESSURE;

        if (src == DLSModulator.CONN_SRC_CC91)
            return new ModelIdentifier("midi_cc", "91", 0);
        if (src == DLSModulator.CONN_SRC_CC93)
            return new ModelIdentifier("midi_cc", "93", 0);

        return null;
    }

    private ModelConnectionBlock convertToModel(DLSModulator mod) {
        ModelIdentifier source = convertToModelSrc(mod.getSource());
        ModelIdentifier control = convertToModelSrc(mod.getControl());
        ModelIdentifier destination_id =
                convertToModelDest(mod.getDestination());

        int scale = mod.getScale();
        double f_scale;
        if (scale == Integer.MIN_VALUE)
            f_scale = Double.NEGATIVE_INFINITY;
        else
            f_scale = scale / 65536.0;

        if (destination_id != null) {
            ModelSource src = null;
            ModelSource ctrl = null;
            ModelConnectionBlock block = new ModelConnectionBlock();
            if (control != null) {
                ModelSource s = new ModelSource();
                if (control == ModelSource.SOURCE_MIDI_PITCH) {
                    ((ModelStandardTransform)s.getTransform()).setPolarity(
                            ModelStandardTransform.POLARITY_BIPOLAR);
                } else if (control == ModelSource.SOURCE_LFO1
                        || control == ModelSource.SOURCE_LFO2) {
                    ((ModelStandardTransform)s.getTransform()).setPolarity(
```

```
                    ModelStandardTransform.POLARITY_BIPOLAR);
            }
            s.setIdentifier(control);
            block.addSource(s);
            ctrl = s;
        }
        if (source != null) {
            ModelSource s = new ModelSource();
            if (source == ModelSource.SOURCE_MIDI_PITCH) {
                ((ModelStandardTransform)s.getTransform()).setPolarity(
                        ModelStandardTransform.POLARITY_BIPOLAR);
            } else if (source == ModelSource.SOURCE_LFO1
                    || source == ModelSource.SOURCE_LFO2) {
                ((ModelStandardTransform)s.getTransform()).setPolarity(
                        ModelStandardTransform.POLARITY_BIPOLAR);
            }
            s.setIdentifier(source);
            block.addSource(s);
            src = s;
        }
        ModelDestination destination = new ModelDestination();
        destination.setIdentifier(destination_id);
        block.setDestination(destination);

        if (mod.getVersion() == 1) {
            //if (mod.getTransform() ==  DLSModulator.CONN_TRN_CONCAVE) {
            //    ((ModelStandardTransform)destination.getTransform())
            //            .setTransform(
            //            ModelStandardTransform.TRANSFORM_CONCAVE);
            //}
            if (mod.getTransform() == DLSModulator.CONN_TRN_CONCAVE) {
                if (src != null) {
                    ((ModelStandardTransform)src.getTransform())
                            .setTransform(
                                ModelStandardTransform.TRANSFORM_CONCAVE);
                    ((ModelStandardTransform)src.getTransform())
                            .setDirection(
                                ModelStandardTransform.DIRECTION_MAX2MIN);
                }
                if (ctrl != null) {
                    ((ModelStandardTransform)ctrl.getTransform())
                            .setTransform(
                                ModelStandardTransform.TRANSFORM_CONCAVE);
                    ((ModelStandardTransform)ctrl.getTransform())
                            .setDirection(
                                ModelStandardTransform.DIRECTION_MAX2MIN);
                }
            }

        } else if (mod.getVersion() == 2) {
            int transform = mod.getTransform();
            int src_transform_invert = (transform >> 15) & 1;
            int src_transform_bipolar = (transform >> 14) & 1;
            int src_transform = (transform >> 10) & 8;
            int ctr_transform_invert = (transform >> 9) & 1;
            int ctr_transform_bipolar = (transform >> 8) & 1;
            int ctr_transform = (transform >> 4) & 8;


            if (src != null) {
                int trans = ModelStandardTransform.TRANSFORM_LINEAR;
                if (src_transform == DLSModulator.CONN_TRN_SWITCH)
```

```
                    trans = ModelStandardTransform.TRANSFORM_SWITCH;
                if (src_transform == DLSModulator.CONN_TRN_CONCAVE)
                    trans = ModelStandardTransform.TRANSFORM_CONCAVE;
                if (src_transform == DLSModulator.CONN_TRN_CONVEX)
                    trans = ModelStandardTransform.TRANSFORM_CONVEX;
                ((ModelStandardTransform)src.getTransform())
                        .setTransform(trans);
                ((ModelStandardTransform)src.getTransform())
                        .setPolarity(src_transform_bipolar == 1);
                ((ModelStandardTransform)src.getTransform())
                        .setDirection(src_transform_invert == 1);

            }

            if (ctrl != null) {
                int trans = ModelStandardTransform.TRANSFORM_LINEAR;
                if (ctr_transform == DLSModulator.CONN_TRN_SWITCH)
                    trans = ModelStandardTransform.TRANSFORM_SWITCH;
                if (ctr_transform == DLSModulator.CONN_TRN_CONCAVE)
                    trans = ModelStandardTransform.TRANSFORM_CONCAVE;
                if (ctr_transform == DLSModulator.CONN_TRN_CONVEX)
                    trans = ModelStandardTransform.TRANSFORM_CONVEX;
                ((ModelStandardTransform)ctrl.getTransform())
                        .setTransform(trans);
                ((ModelStandardTransform)ctrl.getTransform())
                        .setPolarity(ctr_transform_bipolar == 1);
                ((ModelStandardTransform)ctrl.getTransform())
                        .setDirection(ctr_transform_invert == 1);
            }

            /* No output transforms are defined the DLS Level 2
            int out_transform = transform % 8;
            int trans = ModelStandardTransform.TRANSFORM_LINEAR;
            if (out_transform == DLSModulator.CONN_TRN_SWITCH)
                trans = ModelStandardTransform.TRANSFORM_SWITCH;
            if (out_transform == DLSModulator.CONN_TRN_CONCAVE)
                trans = ModelStandardTransform.TRANSFORM_CONCAVE;
            if (out_transform == DLSModulator.CONN_TRN_CONVEX)
                trans = ModelStandardTransform.TRANSFORM_CONVEX;
            if (ctrl != null) {
                ((ModelStandardTransform)destination.getTransform())
                        .setTransform(trans);
            }
            */

        }

        block.setScale(f_scale);

        return block;
    }

    return null;
}

public ModelPerformer[] getPerformers() {
    List<ModelPerformer> performers = new ArrayList<ModelPerformer>();

    Map<String, DLSModulator> modmap = new HashMap<String, DLSModulator>();
    for (DLSModulator mod: getModulators()) {
        modmap.put(mod.getSource() + "x" + mod.getControl() + "=" +
                mod.getDestination(), mod);
```

```
        }

        Map<String, DLSModulator> insmodmap =
                new HashMap<String, DLSModulator>();

        for (DLSRegion zone: regions) {
            ModelPerformer performer = new ModelPerformer();
            performer.setName(zone.getSample().getName());
            performer.setSelfNonExclusive((zone.getFusoptions() &
                    DLSRegion.OPTION_SELFNONEXCLUSIVE) != 0);
            performer.setExclusiveClass(zone.getExclusiveClass());
            performer.setKeyFrom(zone.getKeyfrom());
            performer.setKeyTo(zone.getKeyto());
            performer.setVelFrom(zone.getVelfrom());
            performer.setVelTo(zone.getVelto());

            insmodmap.clear();
            insmodmap.putAll(modmap);
            for (DLSModulator mod: zone.getModulators()) {
                insmodmap.put(mod.getSource() + "x" + mod.getControl() + "=" +
                        mod.getDestination(), mod);
            }

            List<ModelConnectionBlock> blocks = performer.getConnectionBlocks();
            for (DLSModulator mod: insmodmap.values()) {
                ModelConnectionBlock p = convertToModel(mod);
                if (p != null)
                    blocks.add(p);
            }


            DLSSample sample = zone.getSample();
            DLSSampleOptions sampleopt = zone.getSampleoptions();
            if (sampleopt == null)
                sampleopt = sample.getSampleoptions();

            ModelByteBuffer buff = sample.getDataBuffer();

            float pitchcorrection = (-sampleopt.unitynote * 100) +
                    sampleopt.finetune;

            ModelByteBufferWavetable osc = new ModelByteBufferWavetable(buff,
                    sample.getFormat(), pitchcorrection);
            osc.setAttenuation(osc.getAttenuation() / 65536f);
            if (sampleopt.getLoops().size() != 0) {
                DLSSampleLoop loop = sampleopt.getLoops().get(0);
                osc.setLoopStart((int)loop.getStart());
                osc.setLoopLength((int)loop.getLength());
                if (loop.getType() == DLSSampleLoop.LOOP_TYPE_FORWARD)
                    osc.setLoopType(ModelWavetable.LOOP_TYPE_FORWARD);
                if (loop.getType() == DLSSampleLoop.LOOP_TYPE_RELEASE)
                    osc.setLoopType(ModelWavetable.LOOP_TYPE_RELEASE);
                else
                    osc.setLoopType(ModelWavetable.LOOP_TYPE_FORWARD);
            }

            performer.getConnectionBlocks().add(
                    new ModelConnectionBlock(SoftFilter.FILTERTYPE_LP12,
                        new ModelDestination(
                            new ModelIdentifier("filter", "type", 1))));

            performer.getOscillators().add(osc);
```

```
433
434             performers.add(performer);
435
436         }
437
438         return performers.toArray(new ModelPerformer[performers.size()]);
439     }
440
441     public byte[] getGuid() {
442         return guid;
443     }
444
445     public void setGuid(byte[] guid) {
446         this.guid = guid;
447     }
448 }
```

24 com/sun/media/sound/DLSModulator.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * This class is used to store modulator/artiuclation data.
 * A modulator connects one synthesizer source to
 * a destination. For example a note on velocity
 * can be mapped to the gain of the synthesized voice.
 * It is stored as a "art1" or "art2" chunk inside DLS files.
 *
 * @author Karl Helgason
 */
public class DLSModulator {

    // DLS1 Destinations
    public static final int CONN_DST_NONE = 0x000; // 0
    public static final int CONN_DST_GAIN = 0x001; // cB
    public static final int CONN_DST_PITCH = 0x003; // cent
    public static final int CONN_DST_PAN = 0x004; // 0.1%
    public static final int CONN_DST_LFO_FREQUENCY = 0x104; // cent (default 5 Hz)
    public static final int CONN_DST_LFO_STARTDELAY = 0x105; // timecent
    public static final int CONN_DST_EG1_ATTACKTIME = 0x206; // timecent
    public static final int CONN_DST_EG1_DECAYTIME = 0x207; // timecent
    public static final int CONN_DST_EG1_RELEASETIME = 0x209; // timecent
    public static final int CONN_DST_EG1_SUSTAINLEVEL = 0x20A; // 0.1%
    public static final int CONN_DST_EG2_ATTACKTIME = 0x30A; // timecent
    public static final int CONN_DST_EG2_DECAYTIME = 0x30B; // timecent
    public static final int CONN_DST_EG2_RELEASETIME = 0x30D; // timecent
    public static final int CONN_DST_EG2_SUSTAINLEVEL = 0x30E; // 0.1%
    // DLS2 Destinations
    public static final int CONN_DST_KEYNUMBER = 0x005;
    public static final int CONN_DST_LEFT = 0x010; // 0.1%
    public static final int CONN_DST_RIGHT = 0x011; // 0.1%
    public static final int CONN_DST_CENTER = 0x012; // 0.1%
    public static final int CONN_DST_LEFTREAR = 0x013; // 0.1%
    public static final int CONN_DST_RIGHTREAR = 0x014; // 0.1%
    public static final int CONN_DST_LFE_CHANNEL = 0x015; // 0.1%
```

```
    public static final int CONN_DST_CHORUS = 0x080; // 0.1%
    public static final int CONN_DST_REVERB = 0x081; // 0.1%
    public static final int CONN_DST_VIB_FREQUENCY = 0x114; // cent
    public static final int CONN_DST_VIB_STARTDELAY = 0x115; // dB
    public static final int CONN_DST_EG1_DELAYTIME = 0x20B; // timecent
    public static final int CONN_DST_EG1_HOLDTIME = 0x20C; // timecent
    public static final int CONN_DST_EG1_SHUTDOWNTIME = 0x20D; // timecent
    public static final int CONN_DST_EG2_DELAYTIME = 0x30F; // timecent
    public static final int CONN_DST_EG2_HOLDTIME = 0x310; // timecent
    public static final int CONN_DST_FILTER_CUTOFF = 0x500; // cent
    public static final int CONN_DST_FILTER_Q = 0x501; // dB

    // DLS1 Sources
    public static final int CONN_SRC_NONE = 0x000; // 1
    public static final int CONN_SRC_LFO = 0x001; // linear (sine wave)
    public static final int CONN_SRC_KEYONVELOCITY = 0x002; // ??db or velocity??
    public static final int CONN_SRC_KEYNUMBER = 0x003; // ??cent or keynumber??
    public static final int CONN_SRC_EG1 = 0x004; // linear direct from eg
    public static final int CONN_SRC_EG2 = 0x005; // linear direct from eg
    public static final int CONN_SRC_PITCHWHEEL = 0x006; // linear -1..1
    public static final int CONN_SRC_CC1 = 0x081; // linear 0..1
    public static final int CONN_SRC_CC7 = 0x087; // linear 0..1
    public static final int CONN_SRC_CC10 = 0x08A; // linear 0..1
    public static final int CONN_SRC_CC11 = 0x08B; // linear 0..1
    public static final int CONN_SRC_RPN0 = 0x100; // ?? // Pitch Bend Range
    public static final int CONN_SRC_RPN1 = 0x101; // ?? // Fine Tune
    public static final int CONN_SRC_RPN2 = 0x102; // ?? // Course Tune
    // DLS2 Sources
    public static final int CONN_SRC_POLYPRESSURE = 0x007; // linear 0..1
    public static final int CONN_SRC_CHANNELPRESSURE = 0x008; // linear 0..1
    public static final int CONN_SRC_VIBRATO = 0x009; // linear 0..1
    public static final int CONN_SRC_MONOPRESSURE = 0x00A; // linear 0..1
    public static final int CONN_SRC_CC91 = 0x0DB; // linear 0..1
    public static final int CONN_SRC_CC93 = 0x0DD; // linear 0..1
    // DLS1 Transforms
    public static final int CONN_TRN_NONE = 0x000;
    public static final int CONN_TRN_CONCAVE = 0x001;
    // DLS2 Transforms
    public static final int CONN_TRN_CONVEX = 0x002;
    public static final int CONN_TRN_SWITCH = 0x003;
    public static final int DST_FORMAT_CB = 1;
    public static final int DST_FORMAT_CENT = 1;
    public static final int DST_FORMAT_TIMECENT = 2;
    public static final int DST_FORMAT_PERCENT = 3;
    protected int source;
    protected int control;
    protected int destination;
    protected int transform;
    protected int scale;
    protected int version = 1;

    public int getControl() {
        return control;
    }

    public void setControl(int control) {
        this.control = control;
    }

    public static int getDestinationFormat(int destination) {

        if (destination == CONN_DST_GAIN)
```

```
            return DST_FORMAT_CB;
        if (destination == CONN_DST_PITCH)
            return DST_FORMAT_CENT;
        if (destination == CONN_DST_PAN)
            return DST_FORMAT_PERCENT;

        if (destination == CONN_DST_LFO_FREQUENCY)
            return DST_FORMAT_CENT;
        if (destination == CONN_DST_LFO_STARTDELAY)
            return DST_FORMAT_TIMECENT;

        if (destination == CONN_DST_EG1_ATTACKTIME)
            return DST_FORMAT_TIMECENT;
        if (destination == CONN_DST_EG1_DECAYTIME)
            return DST_FORMAT_TIMECENT;
        if (destination == CONN_DST_EG1_RELEASETIME)
            return DST_FORMAT_TIMECENT;
        if (destination == CONN_DST_EG1_SUSTAINLEVEL)
            return DST_FORMAT_PERCENT;

        if (destination == CONN_DST_EG2_ATTACKTIME)
            return DST_FORMAT_TIMECENT;
        if (destination == CONN_DST_EG2_DECAYTIME)
            return DST_FORMAT_TIMECENT;
        if (destination == CONN_DST_EG2_RELEASETIME)
            return DST_FORMAT_TIMECENT;
        if (destination == CONN_DST_EG2_SUSTAINLEVEL)
            return DST_FORMAT_PERCENT;

        if (destination == CONN_DST_KEYNUMBER)
            return DST_FORMAT_CENT; // NOT SURE WITHOUT DLS 2 SPEC
        if (destination == CONN_DST_LEFT)
            return DST_FORMAT_CB;
        if (destination == CONN_DST_RIGHT)
            return DST_FORMAT_CB;
        if (destination == CONN_DST_CENTER)
            return DST_FORMAT_CB;
        if (destination == CONN_DST_LEFTREAR)
            return DST_FORMAT_CB;
        if (destination == CONN_DST_RIGHTREAR)
            return DST_FORMAT_CB;
        if (destination == CONN_DST_LFE_CHANNEL)
            return DST_FORMAT_CB;
        if (destination == CONN_DST_CHORUS)
            return DST_FORMAT_PERCENT;
        if (destination == CONN_DST_REVERB)
            return DST_FORMAT_PERCENT;

        if (destination == CONN_DST_VIB_FREQUENCY)
            return DST_FORMAT_CENT;
        if (destination == CONN_DST_VIB_STARTDELAY)
            return DST_FORMAT_TIMECENT;

        if (destination == CONN_DST_EG1_DELAYTIME)
            return DST_FORMAT_TIMECENT;
        if (destination == CONN_DST_EG1_HOLDTIME)
            return DST_FORMAT_TIMECENT;
        if (destination == CONN_DST_EG1_SHUTDOWNTIME)
            return DST_FORMAT_TIMECENT;

        if (destination == CONN_DST_EG2_DELAYTIME)
            return DST_FORMAT_TIMECENT;
```

```
185         if (destination == CONN_DST_EG2_HOLDTIME)
186             return DST_FORMAT_TIMECENT;
187 
188         if (destination == CONN_DST_FILTER_CUTOFF)
189             return DST_FORMAT_CENT;
190         if (destination == CONN_DST_FILTER_Q)
191             return DST_FORMAT_CB;
192 
193         return -1;
194     }
195 
196     public static String getDestinationName(int destination) {
197 
198         if (destination == CONN_DST_GAIN)
199             return "gain";
200         if (destination == CONN_DST_PITCH)
201             return "pitch";
202         if (destination == CONN_DST_PAN)
203             return "pan";
204 
205         if (destination == CONN_DST_LFO_FREQUENCY)
206             return "lfo1.freq";
207         if (destination == CONN_DST_LFO_STARTDELAY)
208             return "lfo1.delay";
209 
210         if (destination == CONN_DST_EG1_ATTACKTIME)
211             return "eg1.attack";
212         if (destination == CONN_DST_EG1_DECAYTIME)
213             return "eg1.decay";
214         if (destination == CONN_DST_EG1_RELEASETIME)
215             return "eg1.release";
216         if (destination == CONN_DST_EG1_SUSTAINLEVEL)
217             return "eg1.sustain";
218 
219         if (destination == CONN_DST_EG2_ATTACKTIME)
220             return "eg2.attack";
221         if (destination == CONN_DST_EG2_DECAYTIME)
222             return "eg2.decay";
223         if (destination == CONN_DST_EG2_RELEASETIME)
224             return "eg2.release";
225         if (destination == CONN_DST_EG2_SUSTAINLEVEL)
226             return "eg2.sustain";
227 
228         if (destination == CONN_DST_KEYNUMBER)
229             return "keynumber";
230         if (destination == CONN_DST_LEFT)
231             return "left";
232         if (destination == CONN_DST_RIGHT)
233             return "right";
234         if (destination == CONN_DST_CENTER)
235             return "center";
236         if (destination == CONN_DST_LEFTREAR)
237             return "leftrear";
238         if (destination == CONN_DST_RIGHTREAR)
239             return "rightrear";
240         if (destination == CONN_DST_LFE_CHANNEL)
241             return "lfe_channel";
242         if (destination == CONN_DST_CHORUS)
243             return "chorus";
244         if (destination == CONN_DST_REVERB)
245             return "reverb";
246 
```

```
        if (destination == CONN_DST_VIB_FREQUENCY)
            return "vib.freq";
        if (destination == CONN_DST_VIB_STARTDELAY)
            return "vib.delay";

        if (destination == CONN_DST_EG1_DELAYTIME)
            return "eg1.delay";
        if (destination == CONN_DST_EG1_HOLDTIME)
            return "eg1.hold";
        if (destination == CONN_DST_EG1_SHUTDOWNTIME)
            return "eg1.shutdown";

        if (destination == CONN_DST_EG2_DELAYTIME)
            return "eg2.delay";
        if (destination == CONN_DST_EG2_HOLDTIME)
            return "eg.2hold";

        if (destination == CONN_DST_FILTER_CUTOFF)
            return "filter.cutoff"; // NOT SURE WITHOUT DLS 2 SPEC
        if (destination == CONN_DST_FILTER_Q)
            return "filter.q"; // NOT SURE WITHOUT DLS 2 SPEC

        return null;
    }

    public static String getSourceName(int source) {

        if (source == CONN_SRC_NONE)
            return "none";
        if (source == CONN_SRC_LFO)
            return "lfo";
        if (source == CONN_SRC_KEYONVELOCITY)
            return "keyonvelocity";
        if (source == CONN_SRC_KEYNUMBER)
            return "keynumber";
        if (source == CONN_SRC_EG1)
            return "eg1";
        if (source == CONN_SRC_EG2)
            return "eg2";
        if (source == CONN_SRC_PITCHWHEEL)
            return "pitchweel";
        if (source == CONN_SRC_CC1)
            return "cc1";
        if (source == CONN_SRC_CC7)
            return "cc7";
        if (source == CONN_SRC_CC10)
            return "c10";
        if (source == CONN_SRC_CC11)
            return "cc11";

        if (source == CONN_SRC_POLYPRESSURE)
            return "polypressure";
        if (source == CONN_SRC_CHANNELPRESSURE)
            return "channelpressure";
        if (source == CONN_SRC_VIBRATO)
            return "vibrato";
        if (source == CONN_SRC_MONOPRESSURE)
            return "monopressure";
        if (source == CONN_SRC_CC91)
            return "cc91";
        if (source == CONN_SRC_CC93)
            return "cc93";
```

```
        return null;
    }

    public int getDestination() {
        return destination;
    }

    public void setDestination(int destination) {
        this.destination = destination;
    }

    public int getScale() {
        return scale;
    }

    public void setScale(int scale) {
        this.scale = scale;
    }

    public int getSource() {
        return source;
    }

    public void setSource(int source) {
        this.source = source;
    }

    public int getVersion() {
        return version;
    }

    public void setVersion(int version) {
        this.version = version;
    }

    public int getTransform() {
        return transform;
    }

    public void setTransform(int transform) {
        this.transform = transform;
    }
}
```

25 com/sun/media/sound/DLSRegion.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

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

/**
 * This class is used to store region parts for instrument.
 * A region has a velocity and key range which it response to.
 * And it has a list of modulators/articulators which
 * is used how to synthesize a single voice.
 * It is stored inside a "rgn " List Chunk inside DLS files.
 *
 * @author Karl Helgason
 */
public class DLSRegion {

    public final static int OPTION_SELFNONEXCLUSIVE = 0x0001;
    protected List<DLSModulator> modulators = new ArrayList<DLSModulator>();
    protected int keyfrom;
    protected int keyto;
    protected int velfrom;
    protected int velto;
    protected int options;
    protected int exclusiveClass;
    protected int fusoptions;
    protected int phasegroup;
    protected long channel;
    protected DLSSample sample = null;
    protected DLSSampleOptions sampleoptions;

    public List<DLSModulator> getModulators() {
        return modulators;
    }

    public long getChannel() {
        return channel;
```

```
    }

    public void setChannel(long channel) {
        this.channel = channel;
    }

    public int getExclusiveClass() {
        return exclusiveClass;
    }

    public void setExclusiveClass(int exclusiveClass) {
        this.exclusiveClass = exclusiveClass;
    }

    public int getFusoptions() {
        return fusoptions;
    }

    public void setFusoptions(int fusoptions) {
        this.fusoptions = fusoptions;
    }

    public int getKeyfrom() {
        return keyfrom;
    }

    public void setKeyfrom(int keyfrom) {
        this.keyfrom = keyfrom;
    }

    public int getKeyto() {
        return keyto;
    }

    public void setKeyto(int keyto) {
        this.keyto = keyto;
    }

    public int getOptions() {
        return options;
    }

    public void setOptions(int options) {
        this.options = options;
    }

    public int getPhasegroup() {
        return phasegroup;
    }

    public void setPhasegroup(int phasegroup) {
        this.phasegroup = phasegroup;
    }

    public DLSSample getSample() {
        return sample;
    }

    public void setSample(DLSSample sample) {
        this.sample = sample;
    }

```

```
123     public int getVelfrom() {
124         return velfrom;
125     }
126 
127     public void setVelfrom(int velfrom) {
128         this.velfrom = velfrom;
129     }
130 
131     public int getVelto() {
132         return velto;
133     }
134 
135     public void setVelto(int velto) {
136         this.velto = velto;
137     }
138 
139     public void setModulators(List<DLSModulator> modulators) {
140         this.modulators = modulators;
141     }
142 
143     public DLSSampleOptions getSampleoptions() {
144         return sampleoptions;
145     }
146 
147     public void setSampleoptions(DLSSampleOptions sampleOptions) {
148         this.sampleoptions = sampleOptions;
149     }
150 }
```

26 com/sun/media/sound/DLSSample.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.io.InputStream;
import javax.sound.midi.Soundbank;
import javax.sound.midi.SoundbankResource;
import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;

/**
 * This class is used to store the sample data itself.
 * A sample is encoded as PCM audio stream
 * and in DLS Level 1 files it is always a mono 8/16 bit stream.
 * They are stored just like RIFF WAVE files are stored.
 * It is stored inside a "wave" List Chunk inside DLS files.
 *
 * @author Karl Helgason
 */
public class DLSSample extends SoundbankResource {

    protected byte[] guid = null;
    protected DLSInfo info = new DLSInfo();
    protected DLSSampleOptions sampleoptions;
    protected ModelByteBuffer data;
    protected AudioFormat format;

    public DLSSample(Soundbank soundBank) {
        super(soundBank, null, AudioInputStream.class);
    }

    public DLSSample() {
        super(null, null, AudioInputStream.class);
    }

    public DLSInfo getInfo() {
        return info;
    }
```

```
61
62      public Object getData() {
63          AudioFormat format = getFormat();
64
65          InputStream is = data.getInputStream();
66          if (is == null)
67              return null;
68          return new AudioInputStream(is, format, data.capacity());
69      }
70
71      public ModelByteBuffer getDataBuffer() {
72          return data;
73      }
74
75      public AudioFormat getFormat() {
76          return format;
77      }
78
79      public void setFormat(AudioFormat format) {
80          this.format = format;
81      }
82
83      public void setData(ModelByteBuffer data) {
84          this.data = data;
85      }
86
87      public void setData(byte[] data) {
88          this.data = new ModelByteBuffer(data);
89      }
90
91      public void setData(byte[] data, int offset, int length) {
92          this.data = new ModelByteBuffer(data, offset, length);
93      }
94
95      public String getName() {
96          return info.name;
97      }
98
99      public void setName(String name) {
100         info.name = name;
101     }
102
103     public DLSSampleOptions getSampleoptions() {
104         return sampleoptions;
105     }
106
107     public void setSampleoptions(DLSSampleOptions sampleOptions) {
108         this.sampleoptions = sampleOptions;
109     }
110
111     public String toString() {
112         return "Sample:␣" + info.name;
113     }
114
115     public byte[] getGuid() {
116         return guid;
117     }
118
119     public void setGuid(byte[] guid) {
120         this.guid = guid;
121     }
122 }
```

27 com/sun/media/sound/DLSSampleLoop.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * This class is used to store loop points inside DLSSampleOptions class.
 *
 * @author Karl Helgason
 */
public class DLSSampleLoop {

    public final static int LOOP_TYPE_FORWARD = 0;
    public final static int LOOP_TYPE_RELEASE = 1;
    protected long type;
    protected long start;
    protected long length;

    public long getLength() {
        return length;
    }

    public void setLength(long length) {
        this.length = length;
    }

    public long getStart() {
        return start;
    }

    public void setStart(long start) {
        this.start = start;
    }

    public long getType() {
        return type;
    }

    public void setType(long type) {
```

```
61          this.type = type;
62      }
63  }
```

28 com/sun/media/sound/DLSSampleOptions.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

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

/**
 * This class stores options how to playback sampled data like pitch/tuning,
 * attenuation and loops.
 * It is stored as a "wsmp" chunk inside DLS files.
 *
 * @author Karl Helgason
 */
public class DLSSampleOptions {

    protected int unitynote;
    protected short finetune;
    protected int attenuation;
    protected long options;
    protected List<DLSSampleLoop> loops = new ArrayList<DLSSampleLoop>();

    public int getAttenuation() {
        return attenuation;
    }

    public void setAttenuation(int attenuation) {
        this.attenuation = attenuation;
    }

    public short getFinetune() {
        return finetune;
    }

    public void setFinetune(short finetune) {
        this.finetune = finetune;
    }

```

```
    public List<DLSSampleLoop> getLoops() {
        return loops;
    }

    public long getOptions() {
        return options;
    }

    public void setOptions(long options) {
        this.options = options;
    }

    public int getUnitynote() {
        return unitynote;
    }

    public void setUnitynote(int unitynote) {
        this.unitynote = unitynote;
    }
}
```

29 com/sun/media/sound/DLSSoundbank.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.URL;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Stack;

import javax.sound.midi.Instrument;
import javax.sound.midi.Patch;
import javax.sound.midi.Soundbank;
import javax.sound.midi.SoundbankResource;
import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.AudioFormat.Encoding;

/**
 * A DLS Level 1 and Level 2 soundbank reader (from files/url/streams).
 *
 * @author Karl Helgason
 */
public class DLSSoundbank implements Soundbank {

    static private class DLSID {
        long i1;
        int s1;
        int s2;
        int x1;
```

```
        int x2;
        int x3;
        int x4;
        int x5;
        int x6;
        int x7;
        int x8;

        private DLSID() {
        }

        public DLSID(long i1, int s1, int s2, int x1, int x2, int x3, int x4,
                int x5, int x6, int x7, int x8) {
            this.i1 = i1;
            this.s1 = s1;
            this.s2 = s2;
            this.x1 = x1;
            this.x2 = x2;
            this.x3 = x3;
            this.x4 = x4;
            this.x5 = x5;
            this.x6 = x6;
            this.x7 = x7;
            this.x8 = x8;
        }

        public static DLSID read(RIFFReader riff) throws IOException {
            DLSID d = new DLSID();
            d.i1 = riff.readUnsignedInt();
            d.s1 = riff.readUnsignedShort();
            d.s2 = riff.readUnsignedShort();
            d.x1 = riff.readUnsignedByte();
            d.x2 = riff.readUnsignedByte();
            d.x3 = riff.readUnsignedByte();
            d.x4 = riff.readUnsignedByte();
            d.x5 = riff.readUnsignedByte();
            d.x6 = riff.readUnsignedByte();
            d.x7 = riff.readUnsignedByte();
            d.x8 = riff.readUnsignedByte();
            return d;
        }

        public int hashCode() {
            return (int)i1;
        }

        public boolean equals(Object obj) {
            if (!(obj instanceof DLSID)) {
                return false;
            }
            DLSID t = (DLSID) obj;
            return i1 == t.i1 && s1 == t.s1 && s2 == t.s2
                && x1 == t.x1 && x2 == t.x2 && x3 == t.x3 && x4 == t.x4
                && x5 == t.x5 && x6 == t.x6 && x7 == t.x7 && x8 == t.x8;
        }
    }

    /** X = X & Y */
    private static final int DLS_CDL_AND = 0x0001;
    /** X = X | Y */
    private static final int DLS_CDL_OR = 0x0002;
    /** X = X ^ Y */
```

```
    private static final int DLS_CDL_XOR = 0x0003;
    /** X = X + Y */
    private static final int DLS_CDL_ADD = 0x0004;
    /** X = X - Y */
    private static final int DLS_CDL_SUBTRACT = 0x0005;
    /** X = X * Y */
    private static final int DLS_CDL_MULTIPLY = 0x0006;
    /** X = X / Y */
    private static final int DLS_CDL_DIVIDE = 0x0007;
    /** X = X && Y */
    private static final int DLS_CDL_LOGICAL_AND = 0x0008;
    /** X = X || Y */
    private static final int DLS_CDL_LOGICAL_OR = 0x0009;
    /** X = (X < Y) */
    private static final int DLS_CDL_LT = 0x000A;
    /** X = (X <= Y) */
    private static final int DLS_CDL_LE = 0x000B;
    /** X = (X > Y) */
    private static final int DLS_CDL_GT = 0x000C;
    /** X = (X >= Y) */
    private static final int DLS_CDL_GE = 0x000D;
    /** X = (X == Y) */
    private static final int DLS_CDL_EQ = 0x000E;
    /** X = !X */
    private static final int DLS_CDL_NOT = 0x000F;
    /** 32-bit constant */
    private static final int DLS_CDL_CONST = 0x0010;
    /** 32-bit value returned from query */
    private static final int DLS_CDL_QUERY = 0x0011;
    /** 32-bit value returned from query */
    private static final int DLS_CDL_QUERYSUPPORTED = 0x0012;

    private static final DLSID DLSID_GMInHardware = new DLSID(0x178f2f24,
            0xc364, 0x11d1, 0xa7, 0x60, 0x00, 0x00, 0xf8, 0x75, 0xac, 0x12);
    private static final DLSID DLSID_GSInHardware = new DLSID(0x178f2f25,
            0xc364, 0x11d1, 0xa7, 0x60, 0x00, 0x00, 0xf8, 0x75, 0xac, 0x12);
    private static final DLSID DLSID_XGInHardware = new DLSID(0x178f2f26,
            0xc364, 0x11d1, 0xa7, 0x60, 0x00, 0x00, 0xf8, 0x75, 0xac, 0x12);
    private static final DLSID DLSID_SupportsDLS1 = new DLSID(0x178f2f27,
            0xc364, 0x11d1, 0xa7, 0x60, 0x00, 0x00, 0xf8, 0x75, 0xac, 0x12);
    private static final DLSID DLSID_SupportsDLS2 = new DLSID(0xf14599e5,
            0x4689, 0x11d2, 0xaf, 0xa6, 0x0, 0xaa, 0x0, 0x24, 0xd8, 0xb6);
    private static final DLSID DLSID_SampleMemorySize = new DLSID(0x178f2f28,
            0xc364, 0x11d1, 0xa7, 0x60, 0x00, 0x00, 0xf8, 0x75, 0xac, 0x12);
    private static final DLSID DLSID_ManufacturersID = new DLSID(0xb03e1181,
            0x8095, 0x11d2, 0xa1, 0xef, 0x0, 0x60, 0x8, 0x33, 0xdb, 0xd8);
    private static final DLSID DLSID_ProductID = new DLSID(0xb03e1182,
            0x8095, 0x11d2, 0xa1, 0xef, 0x0, 0x60, 0x8, 0x33, 0xdb, 0xd8);
    private static final DLSID DLSID_SamplePlaybackRate = new DLSID(0x2a91f713,
            0xa4bf, 0x11d2, 0xbb, 0xdf, 0x0, 0x60, 0x8, 0x33, 0xdb, 0xd8);

    private long major = -1;
    private long minor = -1;

    private DLSInfo info = new DLSInfo();

    private List<DLSInstrument> instruments = new ArrayList<DLSInstrument>();
    private List<DLSSample> samples = new ArrayList<DLSSample>();

    private boolean largeFormat = false;
    private File sampleFile;

```

```
185     public DLSSoundbank() {
186     }
187
188     public DLSSoundbank(URL url) throws IOException {
189         InputStream is = url.openStream();
190         try {
191             readSoundbank(is);
192         } finally {
193             is.close();
194         }
195     }
196
197     public DLSSoundbank(File file) throws IOException {
198         largeFormat = true;
199         sampleFile = file;
200         InputStream is = new FileInputStream(file);
201         try {
202             readSoundbank(is);
203         } finally {
204             is.close();
205         }
206     }
207
208     public DLSSoundbank(InputStream inputstream) throws IOException {
209         readSoundbank(inputstream);
210     }
211
212     private void readSoundbank(InputStream inputstream) throws IOException {
213         RIFFReader riff = new RIFFReader(inputstream);
214         if (!riff.getFormat().equals("RIFF")) {
215             throw new RIFFInvalidFormatException(
216                     "Input␣stream␣is␣not␣a␣valid␣RIFF␣stream!");
217         }
218         if (!riff.getType().equals("DLS␣")) {
219             throw new RIFFInvalidFormatException(
220                     "Input␣stream␣is␣not␣a␣valid␣DLS␣soundbank!");
221         }
222         while (riff.hasNextChunk()) {
223             RIFFReader chunk = riff.nextChunk();
224             if (chunk.getFormat().equals("LIST")) {
225                 if (chunk.getType().equals("INFO"))
226                     readInfoChunk(chunk);
227                 if (chunk.getType().equals("lins"))
228                     readLinsChunk(chunk);
229                 if (chunk.getType().equals("wvpl"))
230                     readWvplChunk(chunk);
231             } else {
232                 if (chunk.getFormat().equals("cdl␣")) {
233                     if (!readCdlChunk(chunk)) {
234                         throw new RIFFInvalidFormatException(
235                                 "DLS␣file␣isn't␣supported!");
236                     }
237                 }
238                 if (chunk.getFormat().equals("colh")) {
239                     // skipped because we will load the entire bank into memory
240                     // long instrumentcount = chunk.readUnsignedInt();
241                     // System.out.println("instrumentcount = "+ instrumentcount);
242                 }
243                 if (chunk.getFormat().equals("ptbl")) {
244                     // Pool Table Chunk
245                     // skipped because we will load the entire bank into memory
246                 }
```

```
                if (chunk.getFormat().equals("vers")) {
                    major = chunk.readUnsignedInt();
                    minor = chunk.readUnsignedInt();
                }
            }
        }

        for (Map.Entry<DLSRegion, Long> entry : temp_rgnassign.entrySet()) {
            entry.getKey().sample = samples.get((int)entry.getValue().longValue());
        }

        temp_rgnassign = null;
    }

    private boolean cdlIsQuerySupported(DLSID uuid) {
        return uuid.equals(DLSID_GMInHardware)
            || uuid.equals(DLSID_GSInHardware)
            || uuid.equals(DLSID_XGInHardware)
            || uuid.equals(DLSID_SupportsDLS1)
            || uuid.equals(DLSID_SupportsDLS2)
            || uuid.equals(DLSID_SampleMemorySize)
            || uuid.equals(DLSID_ManufacturersID)
            || uuid.equals(DLSID_ProductID)
            || uuid.equals(DLSID_SamplePlaybackRate);
    }

    private long cdlQuery(DLSID uuid) {
        if (uuid.equals(DLSID_GMInHardware))
            return 1;
        if (uuid.equals(DLSID_GSInHardware))
            return 0;
        if (uuid.equals(DLSID_XGInHardware))
            return 0;
        if (uuid.equals(DLSID_SupportsDLS1))
            return 1;
        if (uuid.equals(DLSID_SupportsDLS2))
            return 1;
        if (uuid.equals(DLSID_SampleMemorySize))
            return Runtime.getRuntime().totalMemory();
        if (uuid.equals(DLSID_ManufacturersID))
            return 0;
        if (uuid.equals(DLSID_ProductID))
            return 0;
        if (uuid.equals(DLSID_SamplePlaybackRate))
            return 44100;
        return 0;
    }


    // Reading cdl-ck Chunk
    // "cdl " chunk can only appear inside : DLS,lart,lar2,rgn,rgn2
    private boolean readCdlChunk(RIFFReader riff) throws IOException {

        DLSID uuid;
        long x;
        long y;
        Stack<Long> stack = new Stack<Long>();

        while (riff.available() != 0) {
            int opcode = riff.readUnsignedShort();
            switch (opcode) {
            case DLS_CDL_AND:
```

```
                x = stack.pop();
                y = stack.pop();
                stack.push(Long.valueOf(((x != 0) && (y != 0)) ? 1 : 0));
                break;
            case DLS_CDL_OR:
                x = stack.pop();
                y = stack.pop();
                stack.push(Long.valueOf(((x != 0) || (y != 0)) ? 1 : 0));
                break;
            case DLS_CDL_XOR:
                x = stack.pop();
                y = stack.pop();
                stack.push(Long.valueOf(((x != 0) ^ (y != 0)) ? 1 : 0));
                break;
            case DLS_CDL_ADD:
                x = stack.pop();
                y = stack.pop();
                stack.push(Long.valueOf(x + y));
                break;
            case DLS_CDL_SUBTRACT:
                x = stack.pop();
                y = stack.pop();
                stack.push(Long.valueOf(x - y));
                break;
            case DLS_CDL_MULTIPLY:
                x = stack.pop();
                y = stack.pop();
                stack.push(Long.valueOf(x * y));
                break;
            case DLS_CDL_DIVIDE:
                x = stack.pop();
                y = stack.pop();
                stack.push(Long.valueOf(x / y));
                break;
            case DLS_CDL_LOGICAL_AND:
                x = stack.pop();
                y = stack.pop();
                stack.push(Long.valueOf(((x != 0) && (y != 0)) ? 1 : 0));
                break;
            case DLS_CDL_LOGICAL_OR:
                x = stack.pop();
                y = stack.pop();
                stack.push(Long.valueOf(((x != 0) || (y != 0)) ? 1 : 0));
                break;
            case DLS_CDL_LT:
                x = stack.pop();
                y = stack.pop();
                stack.push(Long.valueOf((x < y) ? 1 : 0));
                break;
            case DLS_CDL_LE:
                x = stack.pop();
                y = stack.pop();
                stack.push(Long.valueOf((x <= y) ? 1 : 0));
                break;
            case DLS_CDL_GT:
                x = stack.pop();
                y = stack.pop();
                stack.push(Long.valueOf((x > y) ? 1 : 0));
                break;
            case DLS_CDL_GE:
                x = stack.pop();
                y = stack.pop();
```

```
                stack.push(Long.valueOf((x >= y) ? 1 : 0));
                break;
            case DLS_CDL_EQ:
                x = stack.pop();
                y = stack.pop();
                stack.push(Long.valueOf((x == y) ? 1 : 0));
                break;
            case DLS_CDL_NOT:
                x = stack.pop();
                y = stack.pop();
                stack.push(Long.valueOf((x == 0) ? 1 : 0));
                break;
            case DLS_CDL_CONST:
                stack.push(Long.valueOf(riff.readUnsignedInt()));
                break;
            case DLS_CDL_QUERY:
                uuid = DLSID.read(riff);
                stack.push(cdlQuery(uuid));
                break;
            case DLS_CDL_QUERYSUPPORTED:
                uuid = DLSID.read(riff);
                stack.push(Long.valueOf(cdlIsQuerySupported(uuid) ? 1 : 0));
                break;
            default:
                break;
            }
        }
        if (stack.isEmpty())
            return false;

        return stack.pop() == 1;
    }

    private void readInfoChunk(RIFFReader riff) throws IOException {
        info.name = null;
        while (riff.hasNextChunk()) {
            RIFFReader chunk = riff.nextChunk();
            String format = chunk.getFormat();
            if (format.equals("INAM"))
                info.name = chunk.readString(chunk.available());
            else if (format.equals("ICRD"))
                info.creationDate = chunk.readString(chunk.available());
            else if (format.equals("IENG"))
                info.engineers = chunk.readString(chunk.available());
            else if (format.equals("IPRD"))
                info.product = chunk.readString(chunk.available());
            else if (format.equals("ICOP"))
                info.copyright = chunk.readString(chunk.available());
            else if (format.equals("ICMT"))
                info.comments = chunk.readString(chunk.available());
            else if (format.equals("ISFT"))
                info.tools = chunk.readString(chunk.available());
            else if (format.equals("IARL"))
                info.archival_location = chunk.readString(chunk.available());
            else if (format.equals("IART"))
                info.artist = chunk.readString(chunk.available());
            else if (format.equals("ICMS"))
                info.commissioned = chunk.readString(chunk.available());
            else if (format.equals("IGNR"))
                info.genre = chunk.readString(chunk.available());
            else if (format.equals("IKEY"))
                info.keywords = chunk.readString(chunk.available());
```

```
            else if (format.equals("IMED"))
                info.medium = chunk.readString(chunk.available());
            else if (format.equals("ISBJ"))
                info.subject = chunk.readString(chunk.available());
            else if (format.equals("ISRC"))
                info.source = chunk.readString(chunk.available());
            else if (format.equals("ISRF"))
                info.source_form = chunk.readString(chunk.available());
            else if (format.equals("ITCH"))
                info.technician = chunk.readString(chunk.available());
        }
    }

    private void readLinsChunk(RIFFReader riff) throws IOException {
        while (riff.hasNextChunk()) {
            RIFFReader chunk = riff.nextChunk();
            if (chunk.getFormat().equals("LIST")) {
                if (chunk.getType().equals("ins "))
                    readInsChunk(chunk);
            }
        }
    }

    private void readInsChunk(RIFFReader riff) throws IOException {
        DLSInstrument instrument = new DLSInstrument(this);

        while (riff.hasNextChunk()) {
            RIFFReader chunk = riff.nextChunk();
            String format = chunk.getFormat();
            if (format.equals("LIST")) {
                if (chunk.getType().equals("INFO")) {
                    readInsInfoChunk(instrument, chunk);
                }
                if (chunk.getType().equals("lrgn")) {
                    while (chunk.hasNextChunk()) {
                        RIFFReader subchunk = chunk.nextChunk();
                        if (subchunk.getFormat().equals("LIST")) {
                            if (subchunk.getType().equals("rgn ")) {
                                DLSRegion split = new DLSRegion();
                                if (readRgnChunk(split, subchunk))
                                    instrument.getRegions().add(split);
                            }
                            if (subchunk.getType().equals("rgn2")) {
                                // support for DLS level 2 regions
                                DLSRegion split = new DLSRegion();
                                if (readRgnChunk(split, subchunk))
                                    instrument.getRegions().add(split);
                            }
                        }
                    }
                }
                if (chunk.getType().equals("lart")) {
                    List<DLSModulator> modlist = new ArrayList<DLSModulator>();
                    while (chunk.hasNextChunk()) {
                        RIFFReader subchunk = chunk.nextChunk();
                        if (chunk.getFormat().equals("cdl ")) {
                            if (!readCdlChunk(chunk)) {
                                modlist.clear();
                                break;
                            }
                        }
                        if (subchunk.getFormat().equals("art1"))
```

```
                        readArt1Chunk(modlist, subchunk);
                }
                instrument.getModulators().addAll(modlist);
            }
            if (chunk.getType().equals("lar2")) {
                // support for DLS level 2 ART
                List<DLSModulator> modlist = new ArrayList<DLSModulator>();
                while (chunk.hasNextChunk()) {
                    RIFFReader subchunk = chunk.nextChunk();
                    if (chunk.getFormat().equals("cdl ")) {
                        if (!readCdlChunk(chunk)) {
                            modlist.clear();
                            break;
                        }
                    }
                    if (subchunk.getFormat().equals("art2"))
                        readArt2Chunk(modlist, subchunk);
                }
                instrument.getModulators().addAll(modlist);
            }
        } else {
            if (format.equals("dlid")) {
                instrument.guid = new byte[16];
                chunk.readFully(instrument.guid);
            }
            if (format.equals("insh")) {
                chunk.readUnsignedInt(); // Read Region Count - ignored

                int bank = chunk.read();             // LSB
                bank += (chunk.read() & 127) << 7;   // MSB
                chunk.read(); // Read Reserved byte
                int drumins = chunk.read();          // Drum Instrument

                int id = chunk.read() & 127; // Read only first 7 bits
                chunk.read(); // Read Reserved byte
                chunk.read(); // Read Reserved byte
                chunk.read(); // Read Reserved byte

                instrument.bank = bank;
                instrument.preset = (int) id;
                instrument.druminstrument = (drumins & 128) > 0;
                //System.out.println("bank="+bank+" drumkit="+drumkit
                //        +" id="+id);
            }

        }
    }
    instruments.add(instrument);
}

private void readArt1Chunk(List<DLSModulator> modulators, RIFFReader riff)
        throws IOException {
    long size = riff.readUnsignedInt();
    long count = riff.readUnsignedInt();

    if (size - 8 != 0)
        riff.skipBytes(size - 8);

    for (int i = 0; i < count; i++) {
        DLSModulator modulator = new DLSModulator();
        modulator.version = 1;
        modulator.source = riff.readUnsignedShort();
```

```
            modulator.control = riff.readUnsignedShort();
            modulator.destination = riff.readUnsignedShort();
            modulator.transform = riff.readUnsignedShort();
            modulator.scale = riff.readInt();
            modulators.add(modulator);
        }
    }

    private void readArt2Chunk(List<DLSModulator> modulators, RIFFReader riff)
            throws IOException {
        long size = riff.readUnsignedInt();
        long count = riff.readUnsignedInt();

        if (size - 8 != 0)
            riff.skipBytes(size - 8);

        for (int i = 0; i < count; i++) {
            DLSModulator modulator = new DLSModulator();
            modulator.version = 2;
            modulator.source = riff.readUnsignedShort();
            modulator.control = riff.readUnsignedShort();
            modulator.destination = riff.readUnsignedShort();
            modulator.transform = riff.readUnsignedShort();
            modulator.scale = riff.readInt();
            modulators.add(modulator);
        }
    }

    private Map<DLSRegion, Long> temp_rgnassign = new HashMap<DLSRegion, Long>();

    private boolean readRgnChunk(DLSRegion split, RIFFReader riff)
            throws IOException {
        while (riff.hasNextChunk()) {
            RIFFReader chunk = riff.nextChunk();
            String format = chunk.getFormat();
            if (format.equals("LIST")) {
                if (chunk.getType().equals("lart")) {
                    List<DLSModulator> modlist = new ArrayList<DLSModulator>();
                    while (chunk.hasNextChunk()) {
                        RIFFReader subchunk = chunk.nextChunk();
                        if (chunk.getFormat().equals("cdl ")) {
                            if (!readCdlChunk(chunk)) {
                                modlist.clear();
                                break;
                            }
                        }
                        if (subchunk.getFormat().equals("art1"))
                            readArt1Chunk(modlist, subchunk);
                    }
                    split.getModulators().addAll(modlist);
                }
                if (chunk.getType().equals("lar2")) {
                    // support for DLS level 2 ART
                    List<DLSModulator> modlist = new ArrayList<DLSModulator>();
                    while (chunk.hasNextChunk()) {
                        RIFFReader subchunk = chunk.nextChunk();
                        if (chunk.getFormat().equals("cdl ")) {
                            if (!readCdlChunk(chunk)) {
                                modlist.clear();
                                break;
                            }
                        }
```

```
                    if (subchunk.getFormat().equals("art2"))
                        readArt2Chunk(modlist, subchunk);
                }
                split.getModulators().addAll(modlist);
            }
        } else {

            if (format.equals("cdl ")) {
                if (!readCdlChunk(chunk))
                    return false;
            }
            if (format.equals("rgnh")) {
                split.keyfrom = chunk.readUnsignedShort();
                split.keyto = chunk.readUnsignedShort();
                split.velfrom = chunk.readUnsignedShort();
                split.velto = chunk.readUnsignedShort();
                split.options = chunk.readUnsignedShort();
                split.exclusiveClass = chunk.readUnsignedShort();
            }
            if (format.equals("wlnk")) {
                split.fusoptions = chunk.readUnsignedShort();
                split.phasegroup = chunk.readUnsignedShort();
                split.channel = chunk.readUnsignedInt();
                long sampleid = chunk.readUnsignedInt();
                temp_rgnassign.put(split, sampleid);
            }
            if (format.equals("wsmp")) {
                split.sampleoptions = new DLSSampleOptions();
                readWsmpChunk(split.sampleoptions, chunk);
            }
        }
    }
    return true;
}

private void readWsmpChunk(DLSSampleOptions sampleOptions, RIFFReader riff)
        throws IOException {
    long size = riff.readUnsignedInt();
    sampleOptions.unitynote = riff.readUnsignedShort();
    sampleOptions.finetune = riff.readShort();
    sampleOptions.attenuation = riff.readInt();
    sampleOptions.options = riff.readUnsignedInt();
    long loops = riff.readInt();

    if (size > 20)
        riff.skipBytes(size - 20);

    for (int i = 0; i < loops; i++) {
        DLSSampleLoop loop = new DLSSampleLoop();
        long size2 = riff.readUnsignedInt();
        loop.type = riff.readUnsignedInt();
        loop.start = riff.readUnsignedInt();
        loop.length = riff.readUnsignedInt();
        sampleOptions.loops.add(loop);
        if (size2 > 16)
            riff.skipBytes(size2 - 16);
    }
}

private void readInsInfoChunk(DLSInstrument dlsinstrument, RIFFReader riff)
        throws IOException {
    dlsinstrument.info.name = null;
```

```
        while (riff.hasNextChunk()) {
            RIFFReader chunk = riff.nextChunk();
            String format = chunk.getFormat();
            if (format.equals("INAM")) {
                dlsinstrument.info.name = chunk.readString(chunk.available());
            } else if (format.equals("ICRD")) {
                dlsinstrument.info.creationDate =
                        chunk.readString(chunk.available());
            } else if (format.equals("IENG")) {
                dlsinstrument.info.engineers =
                        chunk.readString(chunk.available());
            } else if (format.equals("IPRD")) {
                dlsinstrument.info.product = chunk.readString(chunk.available());
            } else if (format.equals("ICOP")) {
                dlsinstrument.info.copyright =
                        chunk.readString(chunk.available());
            } else if (format.equals("ICMT")) {
                dlsinstrument.info.comments =
                        chunk.readString(chunk.available());
            } else if (format.equals("ISFT")) {
                dlsinstrument.info.tools = chunk.readString(chunk.available());
            } else if (format.equals("IARL")) {
                dlsinstrument.info.archival_location =
                        chunk.readString(chunk.available());
            } else if (format.equals("IART")) {
                dlsinstrument.info.artist = chunk.readString(chunk.available());
            } else if (format.equals("ICMS")) {
                dlsinstrument.info.commissioned =
                        chunk.readString(chunk.available());
            } else if (format.equals("IGNR")) {
                dlsinstrument.info.genre = chunk.readString(chunk.available());
            } else if (format.equals("IKEY")) {
                dlsinstrument.info.keywords =
                        chunk.readString(chunk.available());
            } else if (format.equals("IMED")) {
                dlsinstrument.info.medium = chunk.readString(chunk.available());
            } else if (format.equals("ISBJ")) {
                dlsinstrument.info.subject = chunk.readString(chunk.available());
            } else if (format.equals("ISRC")) {
                dlsinstrument.info.source = chunk.readString(chunk.available());
            } else if (format.equals("ISRF")) {
                dlsinstrument.info.source_form =
                        chunk.readString(chunk.available());
            } else if (format.equals("ITCH")) {
                dlsinstrument.info.technician =
                        chunk.readString(chunk.available());
            }
        }
    }

    private void readWvplChunk(RIFFReader riff) throws IOException {
        while (riff.hasNextChunk()) {
            RIFFReader chunk = riff.nextChunk();
            if (chunk.getFormat().equals("LIST")) {
                if (chunk.getType().equals("wave"))
                    readWaveChunk(chunk);
            }
        }
    }

    private void readWaveChunk(RIFFReader riff) throws IOException {
        DLSSample sample = new DLSSample(this);
```

```
743
744         while (riff.hasNextChunk()) {
745             RIFFReader chunk = riff.nextChunk();
746             String format = chunk.getFormat();
747             if (format.equals("LIST")) {
748                 if (chunk.getType().equals("INFO")) {
749                     readWaveInfoChunk(sample, chunk);
750                 }
751             } else {
752                 if (format.equals("dlid")) {
753                     sample.guid = new byte[16];
754                     chunk.readFully(sample.guid);
755                 }
756
757                 if (format.equals("fmt␣")) {
758                     int sampleformat = chunk.readUnsignedShort();
759                     if (sampleformat != 1 && sampleformat != 3) {
760                         throw new RIFFInvalidDataException(
761                                 "Only␣PCM␣samples␣are␣supported!");
762                     }
763                     int channels = chunk.readUnsignedShort();
764                     long samplerate = chunk.readUnsignedInt();
765                     // bytes per sec
766                     /* long framerate = */ chunk.readUnsignedInt();
767                     // block align, framesize
768                     int framesize = chunk.readUnsignedShort();
769                     int bits = chunk.readUnsignedShort();
770                     AudioFormat audioformat = null;
771                     if (sampleformat == 1) {
772                         if (bits == 8) {
773                             audioformat = new AudioFormat(
774                                     Encoding.PCM_UNSIGNED, samplerate, bits,
775                                     channels, framesize, samplerate, false);
776                         } else {
777                             audioformat = new AudioFormat(
778                                     Encoding.PCM_SIGNED, samplerate, bits,
779                                     channels, framesize, samplerate, false);
780                         }
781                     }
782                     if (sampleformat == 3) {
783                         audioformat = new AudioFormat(
784                                 AudioFloatConverter.PCM_FLOAT, samplerate, bits,
785                                 channels, framesize, samplerate, false);
786                     }
787
788                     sample.format = audioformat;
789                 }
790
791                 if (format.equals("data")) {
792                     if (largeFormat) {
793                         sample.setData(new ModelByteBuffer(sampleFile,
794                                 chunk.getFilePointer(), chunk.available()));
795                     } else {
796                         byte[] buffer = new byte[chunk.available()];
797                         //  chunk.read(buffer);
798                         sample.setData(buffer);
799
800                         int read = 0;
801                         int avail = chunk.available();
802                         while (read != avail) {
803                             if (avail - read > 65536) {
804                                 chunk.readFully(buffer, read, 65536);
```

```
                            read += 65536;
                        } else {
                            chunk.readFully(buffer, read, avail - read);
                            read = avail;
                        }
                    }
                }
            }

            if (format.equals("wsmp")) {
                sample.sampleoptions = new DLSSampleOptions();
                readWsmpChunk(sample.sampleoptions, chunk);
            }
        }
    }

    samples.add(sample);

}

private void readWaveInfoChunk(DLSSample dlssample, RIFFReader riff)
        throws IOException {
    dlssample.info.name = null;
    while (riff.hasNextChunk()) {
        RIFFReader chunk = riff.nextChunk();
        String format = chunk.getFormat();
        if (format.equals("INAM")) {
            dlssample.info.name = chunk.readString(chunk.available());
        } else if (format.equals("ICRD")) {
            dlssample.info.creationDate =
                    chunk.readString(chunk.available());
        } else if (format.equals("IENG")) {
            dlssample.info.engineers = chunk.readString(chunk.available());
        } else if (format.equals("IPRD")) {
            dlssample.info.product = chunk.readString(chunk.available());
        } else if (format.equals("ICOP")) {
            dlssample.info.copyright = chunk.readString(chunk.available());
        } else if (format.equals("ICMT")) {
            dlssample.info.comments = chunk.readString(chunk.available());
        } else if (format.equals("ISFT")) {
            dlssample.info.tools = chunk.readString(chunk.available());
        } else if (format.equals("IARL")) {
            dlssample.info.archival_location =
                    chunk.readString(chunk.available());
        } else if (format.equals("IART")) {
            dlssample.info.artist = chunk.readString(chunk.available());
        } else if (format.equals("ICMS")) {
            dlssample.info.commissioned =
                    chunk.readString(chunk.available());
        } else if (format.equals("IGNR")) {
            dlssample.info.genre = chunk.readString(chunk.available());
        } else if (format.equals("IKEY")) {
            dlssample.info.keywords = chunk.readString(chunk.available());
        } else if (format.equals("IMED")) {
            dlssample.info.medium = chunk.readString(chunk.available());
        } else if (format.equals("ISBJ")) {
            dlssample.info.subject = chunk.readString(chunk.available());
        } else if (format.equals("ISRC")) {
            dlssample.info.source = chunk.readString(chunk.available());
        } else if (format.equals("ISRF")) {
            dlssample.info.source_form = chunk.readString(chunk.available());
        } else if (format.equals("ITCH")) {
```

```
                dlssample.info.technician = chunk.readString(chunk.available());
            }
        }
    }

    public void save(String name) throws IOException {
        writeSoundbank(new RIFFWriter(name, "DLS "));
    }

    public void save(File file) throws IOException {
        writeSoundbank(new RIFFWriter(file, "DLS "));
    }

    public void save(OutputStream out) throws IOException {
        writeSoundbank(new RIFFWriter(out, "DLS "));
    }

    private void writeSoundbank(RIFFWriter writer) throws IOException {
        RIFFWriter colh_chunk = writer.writeChunk("colh");
        colh_chunk.writeUnsignedInt(instruments.size());

        if (major != -1 && minor != -1) {
            RIFFWriter vers_chunk = writer.writeChunk("vers");
            vers_chunk.writeUnsignedInt(major);
            vers_chunk.writeUnsignedInt(minor);
        }

        writeInstruments(writer.writeList("lins"));

        RIFFWriter ptbl = writer.writeChunk("ptbl");
        ptbl.writeUnsignedInt(8);
        ptbl.writeUnsignedInt(samples.size());
        long ptbl_offset = writer.getFilePointer();
        for (int i = 0; i < samples.size(); i++)
            ptbl.writeUnsignedInt(0);

        RIFFWriter wvpl = writer.writeList("wvpl");
        long off = wvpl.getFilePointer();
        List<Long> offsettable = new ArrayList<Long>();
        for (DLSSample sample : samples) {
            offsettable.add(Long.valueOf(wvpl.getFilePointer() - off));
            writeSample(wvpl.writeList("wave"), sample);
        }

        // small cheat, we are going to rewrite data back in wvpl
        long bak = writer.getFilePointer();
        writer.seek(ptbl_offset);
        writer.setWriteOverride(true);
        for (Long offset : offsettable)
            writer.writeUnsignedInt(offset.longValue());
        writer.setWriteOverride(false);
        writer.seek(bak);

        writeInfo(writer.writeList("INFO"), info);

        writer.close();
    }

    private void writeSample(RIFFWriter writer, DLSSample sample)
            throws IOException {

        AudioFormat audioformat = sample.getFormat();
```

```
929
930         Encoding encoding = audioformat.getEncoding();
931         float sampleRate = audioformat.getSampleRate();
932         int sampleSizeInBits = audioformat.getSampleSizeInBits();
933         int channels = audioformat.getChannels();
934         int frameSize = audioformat.getFrameSize();
935         float frameRate = audioformat.getFrameRate();
936         boolean bigEndian = audioformat.isBigEndian();
937 
938         boolean convert_needed = false;
939 
940         if (audioformat.getSampleSizeInBits() == 8) {
941             if (!encoding.equals(Encoding.PCM_UNSIGNED)) {
942                 encoding = Encoding.PCM_UNSIGNED;
943                 convert_needed = true;
944             }
945         } else {
946             if (!encoding.equals(Encoding.PCM_SIGNED)) {
947                 encoding = Encoding.PCM_SIGNED;
948                 convert_needed = true;
949             }
950             if (bigEndian) {
951                 bigEndian = false;
952                 convert_needed = true;
953             }
954         }
955 
956         if (convert_needed) {
957             audioformat = new AudioFormat(encoding, sampleRate,
958                     sampleSizeInBits, channels, frameSize, frameRate, bigEndian);
959         }
960 
961         // fmt
962         RIFFWriter fmt_chunk = writer.writeChunk("fmt ");
963         int sampleformat = 0;
964         if (audioformat.getEncoding().equals(Encoding.PCM_UNSIGNED))
965             sampleformat = 1;
966         else if (audioformat.getEncoding().equals(Encoding.PCM_SIGNED))
967             sampleformat = 1;
968         else if (audioformat.getEncoding().equals(AudioFloatConverter.PCM_FLOAT))
969             sampleformat = 3;
970 
971         fmt_chunk.writeUnsignedShort(sampleformat);
972         fmt_chunk.writeUnsignedShort(audioformat.getChannels());
973         fmt_chunk.writeUnsignedInt((long) audioformat.getSampleRate());
974         long srate = ((long)audioformat.getFrameRate())*audioformat.getFrameSize();
975         fmt_chunk.writeUnsignedInt(srate);
976         fmt_chunk.writeUnsignedShort(audioformat.getFrameSize());
977         fmt_chunk.writeUnsignedShort(audioformat.getSampleSizeInBits());
978         fmt_chunk.write(0);
979         fmt_chunk.write(0);
980 
981         writeSampleOptions(writer.writeChunk("wsmp"), sample.sampleoptions);
982 
983         if (convert_needed) {
984             RIFFWriter data_chunk = writer.writeChunk("data");
985             AudioInputStream stream = AudioSystem.getAudioInputStream(
986                     audioformat, (AudioInputStream)sample.getData());
987             byte[] buff = new byte[1024];
988             int ret;
989             while ((ret = stream.read(buff)) != -1) {
990                 data_chunk.write(buff, 0, ret);
```

```
            }
        } else {
            RIFFWriter data_chunk = writer.writeChunk("data");
            ModelByteBuffer databuff = sample.getDataBuffer();
            databuff.writeTo(data_chunk);
            /*
            data_chunk.write(databuff.array(),
            databuff.arrayOffset(),
            databuff.capacity());
             */
        }

        writeInfo(writer.writeList("INFO"), sample.info);
    }

    private void writeInstruments(RIFFWriter writer) throws IOException {
        for (DLSInstrument instrument : instruments) {
            writeInstrument(writer.writeList("ins "), instrument);
        }
    }

    private void writeInstrument(RIFFWriter writer, DLSInstrument instrument)
            throws IOException {

        int art1_count = 0;
        int art2_count = 0;
        for (DLSModulator modulator : instrument.getModulators()) {
            if (modulator.version == 1)
                art1_count++;
            if (modulator.version == 2)
                art2_count++;
        }
        for (DLSRegion region : instrument.regions) {
            for (DLSModulator modulator : region.getModulators()) {
                if (modulator.version == 1)
                    art1_count++;
                if (modulator.version == 2)
                    art2_count++;
            }
        }

        int version = 1;
        if (art2_count > 0)
            version = 2;

        RIFFWriter insh_chunk = writer.writeChunk("insh");
        insh_chunk.writeUnsignedInt(instrument.getRegions().size());
        insh_chunk.writeUnsignedInt(instrument.bank +
                (instrument.druminstrument ? 2147483648L : 0));
        insh_chunk.writeUnsignedInt(instrument.preset);

        RIFFWriter lrgn = writer.writeList("lrgn");
        for (DLSRegion region: instrument.regions)
            writeRegion(lrgn, region, version);

        writeArticulators(writer, instrument.getModulators());

        writeInfo(writer.writeList("INFO"), instrument.info);

    }

    private void writeArticulators(RIFFWriter writer,
```

```
            List<DLSModulator> modulators) throws IOException {
        int art1_count = 0;
        int art2_count = 0;
        for (DLSModulator modulator : modulators) {
            if (modulator.version == 1)
                art1_count++;
            if (modulator.version == 2)
                art2_count++;
        }
        if (art1_count > 0) {
            RIFFWriter lar1 = writer.writeList("lart");
            RIFFWriter art1 = lar1.writeChunk("art1");
            art1.writeUnsignedInt(8);
            art1.writeUnsignedInt(art1_count);
            for (DLSModulator modulator : modulators) {
                if (modulator.version == 1) {
                    art1.writeUnsignedShort(modulator.source);
                    art1.writeUnsignedShort(modulator.control);
                    art1.writeUnsignedShort(modulator.destination);
                    art1.writeUnsignedShort(modulator.transform);
                    art1.writeInt(modulator.scale);
                }
            }
        }
        if (art2_count > 0) {
            RIFFWriter lar2 = writer.writeList("lar2");
            RIFFWriter art2 = lar2.writeChunk("art2");
            art2.writeUnsignedInt(8);
            art2.writeUnsignedInt(art2_count);
            for (DLSModulator modulator : modulators) {
                if (modulator.version == 2) {
                    art2.writeUnsignedShort(modulator.source);
                    art2.writeUnsignedShort(modulator.control);
                    art2.writeUnsignedShort(modulator.destination);
                    art2.writeUnsignedShort(modulator.transform);
                    art2.writeInt(modulator.scale);
                }
            }
        }
    }

    private void writeRegion(RIFFWriter writer, DLSRegion region, int version)
            throws IOException {
        RIFFWriter rgns = null;
        if (version == 1)
            rgns = writer.writeList("rgn␣");
        if (version == 2)
            rgns = writer.writeList("rgn2");
        if (rgns == null)
            return;

        RIFFWriter rgnh = rgns.writeChunk("rgnh");
        rgnh.writeUnsignedShort(region.keyfrom);
        rgnh.writeUnsignedShort(region.keyto);
        rgnh.writeUnsignedShort(region.velfrom);
        rgnh.writeUnsignedShort(region.velto);
        rgnh.writeUnsignedShort(region.options);
        rgnh.writeUnsignedShort(region.exclusiveClass);

        if (region.sampleoptions != null)
            writeSampleOptions(rgns.writeChunk("wsmp"), region.sampleoptions);

```

```
        if (region.sample != null) {
            if (samples.indexOf(region.sample) != -1) {
                RIFFWriter wlnk = rgns.writeChunk("wlnk");
                wlnk.writeUnsignedShort(region.fusoptions);
                wlnk.writeUnsignedShort(region.phasegroup);
                wlnk.writeUnsignedInt(region.channel);
                wlnk.writeUnsignedInt(samples.indexOf(region.sample));
            }
        }
        writeArticulators(rgns, region.getModulators());
        rgns.close();
    }

    private void writeSampleOptions(RIFFWriter wsmp,
            DLSSampleOptions sampleoptions) throws IOException {
        wsmp.writeUnsignedInt(20);
        wsmp.writeUnsignedShort(sampleoptions.unitynote);
        wsmp.writeShort(sampleoptions.finetune);
        wsmp.writeInt(sampleoptions.attenuation);
        wsmp.writeUnsignedInt(sampleoptions.options);
        wsmp.writeInt(sampleoptions.loops.size());

        for (DLSSampleLoop loop : sampleoptions.loops) {
            wsmp.writeUnsignedInt(16);
            wsmp.writeUnsignedInt(loop.type);
            wsmp.writeUnsignedInt(loop.start);
            wsmp.writeUnsignedInt(loop.length);
        }
    }

    private void writeInfoStringChunk(RIFFWriter writer,
            String name, String value) throws IOException {
        if (value == null)
            return;
        RIFFWriter chunk = writer.writeChunk(name);
        chunk.writeString(value);
        int len = value.getBytes("ascii").length;
        chunk.write(0);
        len++;
        if (len % 2 != 0)
            chunk.write(0);
    }

    private void writeInfo(RIFFWriter writer, DLSInfo info) throws IOException {
        writeInfoStringChunk(writer, "INAM", info.name);
        writeInfoStringChunk(writer, "ICRD", info.creationDate);
        writeInfoStringChunk(writer, "IENG", info.engineers);
        writeInfoStringChunk(writer, "IPRD", info.product);
        writeInfoStringChunk(writer, "ICOP", info.copyright);
        writeInfoStringChunk(writer, "ICMT", info.comments);
        writeInfoStringChunk(writer, "ISFT", info.tools);
        writeInfoStringChunk(writer, "IARL", info.archival_location);
        writeInfoStringChunk(writer, "IART", info.artist);
        writeInfoStringChunk(writer, "ICMS", info.commissioned);
        writeInfoStringChunk(writer, "IGNR", info.genre);
        writeInfoStringChunk(writer, "IKEY", info.keywords);
        writeInfoStringChunk(writer, "IMED", info.medium);
        writeInfoStringChunk(writer, "ISBJ", info.subject);
        writeInfoStringChunk(writer, "ISRC", info.source);
        writeInfoStringChunk(writer, "ISRF", info.source_form);
        writeInfoStringChunk(writer, "ITCH", info.technician);
    }
```

```
1177
1178    public DLSInfo getInfo() {
1179        return info;
1180    }
1181
1182    public String getName() {
1183        return info.name;
1184    }
1185
1186    public String getVersion() {
1187        return major + "." + minor;
1188    }
1189
1190    public String getVendor() {
1191        return info.engineers;
1192    }
1193
1194    public String getDescription() {
1195        return info.comments;
1196    }
1197
1198    public void setName(String s) {
1199        info.name = s;
1200    }
1201
1202    public void setVendor(String s) {
1203        info.engineers = s;
1204    }
1205
1206    public void setDescription(String s) {
1207        info.comments = s;
1208    }
1209
1210    public SoundbankResource[] getResources() {
1211        SoundbankResource[] resources = new SoundbankResource[samples.size()];
1212        int j = 0;
1213        for (int i = 0; i < samples.size(); i++)
1214            resources[j++] = samples.get(i);
1215        return resources;
1216    }
1217
1218    public DLSInstrument[] getInstruments() {
1219        DLSInstrument[] inslist_array =
1220                instruments.toArray(new DLSInstrument[instruments.size()]);
1221        Arrays.sort(inslist_array, new ModelInstrumentComparator());
1222        return inslist_array;
1223    }
1224
1225    public DLSSample[] getSamples() {
1226        return samples.toArray(new DLSSample[samples.size()]);
1227    }
1228
1229    public Instrument getInstrument(Patch patch) {
1230        int program = patch.getProgram();
1231        int bank = patch.getBank();
1232        boolean percussion = false;
1233        if (patch instanceof ModelPatch)
1234            percussion = ((ModelPatch) patch).isPercussion();
1235        for (Instrument instrument : instruments) {
1236            Patch patch2 = instrument.getPatch();
1237            int program2 = patch2.getProgram();
1238            int bank2 = patch2.getBank();
```

```
            if (program == program2 && bank == bank2) {
                boolean percussion2 = false;
                if (patch2 instanceof ModelPatch)
                    percussion2 = ((ModelPatch) patch2).isPercussion();
                if (percussion == percussion2)
                    return instrument;
            }
        }
        return null;
    }

    public void addResource(SoundbankResource resource) {
        if (resource instanceof DLSInstrument)
            instruments.add((DLSInstrument) resource);
        if (resource instanceof DLSSample)
            samples.add((DLSSample) resource);
    }

    public void removeResource(SoundbankResource resource) {
        if (resource instanceof DLSInstrument)
            instruments.remove((DLSInstrument) resource);
        if (resource instanceof DLSSample)
            samples.remove((DLSSample) resource);
    }

    public void addInstrument(DLSInstrument resource) {
        instruments.add(resource);
    }

    public void removeInstrument(DLSInstrument resource) {
        instruments.remove(resource);
    }

    public long getMajor() {
        return major;
    }

    public void setMajor(long major) {
        this.major = major;
    }

    public long getMinor() {
        return minor;
    }

    public void setMinor(long minor) {
        this.minor = minor;
    }
}
```

30 com/sun/media/sound/DLSSoundbankReader.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */

package com.sun.media.sound;

import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import javax.sound.midi.InvalidMidiDataException;
import javax.sound.midi.Soundbank;
import javax.sound.midi.spi.SoundbankReader;

/**
 * This class is used to connect the DLSSoundBank class
 * to the SoundbankReader SPI interface.
 *
 * @author Karl Helgason
 */
public class DLSSoundbankReader extends SoundbankReader {

    public Soundbank getSoundbank(URL url)
            throws InvalidMidiDataException, IOException {
        try {
            return new DLSSoundbank(url);
        } catch (RIFFInvalidFormatException e) {
            return null;
        } catch(IOException ioe) {
            return null;
        }
    }

    public Soundbank getSoundbank(InputStream stream)
            throws InvalidMidiDataException, IOException {
        try {
            stream.mark(512);
            return new DLSSoundbank(stream);
        } catch (RIFFInvalidFormatException e) {
```

```
61              stream.reset();
62              return null;
63          }
64      }
65
66      public Soundbank getSoundbank(File file)
67              throws InvalidMidiDataException, IOException {
68          try {
69              return new DLSSoundbank(file);
70          } catch (RIFFInvalidFormatException e) {
71              return null;
72          }
73      }
74  }
```

31 com/sun/media/sound/EmergencySoundbank.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.util.Random;

import javax.sound.midi.Patch;
import javax.sound.sampled.AudioFormat;

/**
 * Emergency Soundbank generator.
 * Used when no other default soundbank can be found.
 *
 * @author Karl Helgason
 */
public class EmergencySoundbank {

    private final static String[] general_midi_instruments = {
        "Acoustic Grand Piano",
        "Bright Acoustic Piano",
        "Electric Grand Piano",
        "Honky-tonk Piano",
        "Electric Piano 1",
        "Electric Piano 2",
        "Harpsichord",
        "Clavi",
        "Celesta",
        "Glockenspiel",
        "Music Box",
        "Vibraphone",
        "Marimba",
        "Xylophone",
        "Tubular Bells",
        "Dulcimer",
        "Drawbar Organ",
        "Percussive Organ",
        "Rock Organ",
        "Church Organ",
```

```
        "Reed Organ",
        "Accordion",
        "Harmonica",
        "Tango Accordion",
        "Acoustic Guitar (nylon)",
        "Acoustic Guitar (steel)",
        "Electric Guitar (jazz)",
        "Electric Guitar (clean)",
        "Electric Guitar (muted)",
        "Overdriven Guitar",
        "Distortion Guitar",
        "Guitar harmonics",
        "Acoustic Bass",
        "Electric Bass (finger)",
        "Electric Bass (pick)",
        "Fretless Bass",
        "Slap Bass 1",
        "Slap Bass 2",
        "Synth Bass 1",
        "Synth Bass 2",
        "Violin",
        "Viola",
        "Cello",
        "Contrabass",
        "Tremolo Strings",
        "Pizzicato Strings",
        "Orchestral Harp",
        "Timpani",
        "String Ensemble 1",
        "String Ensemble 2",
        "SynthStrings 1",
        "SynthStrings 2",
        "Choir Aahs",
        "Voice Oohs",
        "Synth Voice",
        "Orchestra Hit",
        "Trumpet",
        "Trombone",
        "Tuba",
        "Muted Trumpet",
        "French Horn",
        "Brass Section",
        "SynthBrass 1",
        "SynthBrass 2",
        "Soprano Sax",
        "Alto Sax",
        "Tenor Sax",
        "Baritone Sax",
        "Oboe",
        "English Horn",
        "Bassoon",
        "Clarinet",
        "Piccolo",
        "Flute",
        "Recorder",
        "Pan Flute",
        "Blown Bottle",
        "Shakuhachi",
        "Whistle",
        "Ocarina",
        "Lead 1 (square)",
        "Lead 2 (sawtooth)",
```

```
        "Lead 3 (calliope)",
        "Lead 4 (chiff)",
        "Lead 5 (charang)",
        "Lead 6 (voice)",
        "Lead 7 (fifths)",
        "Lead 8 (bass + lead)",
        "Pad 1 (new age)",
        "Pad 2 (warm)",
        "Pad 3 (polysynth)",
        "Pad 4 (choir)",
        "Pad 5 (bowed)",
        "Pad 6 (metallic)",
        "Pad 7 (halo)",
        "Pad 8 (sweep)",
        "FX 1 (rain)",
        "FX 2 (soundtrack)",
        "FX 3 (crystal)",
        "FX 4 (atmosphere)",
        "FX 5 (brightness)",
        "FX 6 (goblins)",
        "FX 7 (echoes)",
        "FX 8 (sci-fi)",
        "Sitar",
        "Banjo",
        "Shamisen",
        "Koto",
        "Kalimba",
        "Bag pipe",
        "Fiddle",
        "Shanai",
        "Tinkle Bell",
        "Agogo",
        "Steel Drums",
        "Woodblock",
        "Taiko Drum",
        "Melodic Tom",
        "Synth Drum",
        "Reverse Cymbal",
        "Guitar Fret Noise",
        "Breath Noise",
        "Seashore",
        "Bird Tweet",
        "Telephone Ring",
        "Helicopter",
        "Applause",
        "Gunshot"
    };

    public static SF2Soundbank createSoundbank() throws Exception {
        SF2Soundbank sf2 = new SF2Soundbank();
        sf2.setName("Emergency GM sound set");
        sf2.setVendor("Generated");
        sf2.setDescription("Emergency generated soundbank");

        /*
         *  percussion instruments
         */

        SF2Layer bass_drum = new_bass_drum(sf2);
        SF2Layer snare_drum = new_snare_drum(sf2);
        SF2Layer tom = new_tom(sf2);
        SF2Layer open_hihat = new_open_hihat(sf2);
```

```
        SF2Layer closed_hihat = new_closed_hihat(sf2);
        SF2Layer crash_cymbal = new_crash_cymbal(sf2);
        SF2Layer side_stick = new_side_stick(sf2);

        SF2Layer[] drums = new SF2Layer[128];
        drums[35] = bass_drum;
        drums[36] = bass_drum;
        drums[38] = snare_drum;
        drums[40] = snare_drum;
        drums[41] = tom;
        drums[43] = tom;
        drums[45] = tom;
        drums[47] = tom;
        drums[48] = tom;
        drums[50] = tom;
        drums[42] = closed_hihat;
        drums[44] = closed_hihat;
        drums[46] = open_hihat;
        drums[49] = crash_cymbal;
        drums[51] = crash_cymbal;
        drums[52] = crash_cymbal;
        drums[55] = crash_cymbal;
        drums[57] = crash_cymbal;
        drums[59] = crash_cymbal;

        // Use side_stick for missing drums:
        drums[37] = side_stick;
        drums[39] = side_stick;
        drums[53] = side_stick;
        drums[54] = side_stick;
        drums[56] = side_stick;
        drums[58] = side_stick;
        drums[69] = side_stick;
        drums[70] = side_stick;
        drums[75] = side_stick;
        drums[60] = side_stick;
        drums[61] = side_stick;
        drums[62] = side_stick;
        drums[63] = side_stick;
        drums[64] = side_stick;
        drums[65] = side_stick;
        drums[66] = side_stick;
        drums[67] = side_stick;
        drums[68] = side_stick;
        drums[71] = side_stick;
        drums[72] = side_stick;
        drums[73] = side_stick;
        drums[74] = side_stick;
        drums[76] = side_stick;
        drums[77] = side_stick;
        drums[78] = side_stick;
        drums[79] = side_stick;
        drums[80] = side_stick;
        drums[81] = side_stick;


        SF2Instrument drum_instrument = new SF2Instrument(sf2);
        drum_instrument.setName("Standard Kit");
        drum_instrument.setPatch(new ModelPatch(0, 0, true));
        sf2.addInstrument(drum_instrument);
        for (int i = 0; i < drums.length; i++) {
            if (drums[i] != null) {
```

```
247                 SF2InstrumentRegion region = new SF2InstrumentRegion();
248                 region.setLayer(drums[i]);
249                 region.putBytes(SF2InstrumentRegion.GENERATOR_KEYRANGE,
250                         new byte[]{(byte) i, (byte) i});
251                 drum_instrument.getRegions().add(region);
252             }
253         }
254 
255 
256         /*
257          *  melodic instruments
258          */
259 
260         SF2Layer gpiano = new_gpiano(sf2);
261         SF2Layer gpiano2 = new_gpiano2(sf2);
262         SF2Layer gpiano_hammer = new_piano_hammer(sf2);
263         SF2Layer piano1 = new_piano1(sf2);
264         SF2Layer epiano1 = new_epiano1(sf2);
265         SF2Layer epiano2 = new_epiano2(sf2);
266 
267         SF2Layer guitar = new_guitar1(sf2);
268         SF2Layer guitar_pick = new_guitar_pick(sf2);
269         SF2Layer guitar_dist = new_guitar_dist(sf2);
270         SF2Layer bass1 = new_bass1(sf2);
271         SF2Layer bass2 = new_bass2(sf2);
272         SF2Layer synthbass = new_synthbass(sf2);
273         SF2Layer string2 = new_string2(sf2);
274         SF2Layer orchhit = new_orchhit(sf2);
275         SF2Layer choir = new_choir(sf2);
276         SF2Layer solostring = new_solostring(sf2);
277         SF2Layer organ = new_organ(sf2);
278         SF2Layer ch_organ = new_ch_organ(sf2);
279         SF2Layer bell = new_bell(sf2);
280         SF2Layer flute = new_flute(sf2);
281 
282         SF2Layer timpani = new_timpani(sf2);
283         SF2Layer melodic_toms = new_melodic_toms(sf2);
284         SF2Layer trumpet = new_trumpet(sf2);
285         SF2Layer trombone = new_trombone(sf2);
286         SF2Layer brass_section = new_brass_section(sf2);
287         SF2Layer horn = new_horn(sf2);
288         SF2Layer sax = new_sax(sf2);
289         SF2Layer oboe = new_oboe(sf2);
290         SF2Layer bassoon = new_bassoon(sf2);
291         SF2Layer clarinet = new_clarinet(sf2);
292         SF2Layer reverse_cymbal = new_reverse_cymbal(sf2);
293 
294         SF2Layer defaultsound = piano1;
295 
296         newInstrument(sf2, "Piano", new Patch(0, 0), gpiano, gpiano_hammer);
297         newInstrument(sf2, "Piano", new Patch(0, 1), gpiano2, gpiano_hammer);
298         newInstrument(sf2, "Piano", new Patch(0, 2), piano1);
299         {
300             SF2Instrument ins = newInstrument(sf2, "Honky-tonk␣Piano",
301                     new Patch(0, 3), piano1, piano1);
302             SF2InstrumentRegion region = ins.getRegions().get(0);
303             region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 80);
304             region.putInteger(SF2Region.GENERATOR_FINETUNE, 30);
305             region = ins.getRegions().get(1);
306             region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 30);
307         }
308         newInstrument(sf2, "Rhodes", new Patch(0, 4), epiano2);
```

```
309         newInstrument(sf2, "Rhodes", new Patch(0, 5), epiano2);
310         newInstrument(sf2, "Clavinet", new Patch(0, 6), epiano1);
311         newInstrument(sf2, "Clavinet", new Patch(0, 7), epiano1);
312         newInstrument(sf2, "Rhodes", new Patch(0, 8), epiano2);
313         newInstrument(sf2, "Bell", new Patch(0, 9), bell);
314         newInstrument(sf2, "Bell", new Patch(0, 10), bell);
315         newInstrument(sf2, "Vibraphone", new Patch(0, 11), bell);
316         newInstrument(sf2, "Marimba", new Patch(0, 12), bell);
317         newInstrument(sf2, "Marimba", new Patch(0, 13), bell);
318         newInstrument(sf2, "Bell", new Patch(0, 14), bell);
319         newInstrument(sf2, "Rock Organ", new Patch(0, 15), organ);
320         newInstrument(sf2, "Rock Organ", new Patch(0, 16), organ);
321         newInstrument(sf2, "Perc Organ", new Patch(0, 17), organ);
322         newInstrument(sf2, "Rock Organ", new Patch(0, 18), organ);
323         newInstrument(sf2, "Church Organ", new Patch(0, 19), ch_organ);
324         newInstrument(sf2, "Accordion", new Patch(0, 20), organ);
325         newInstrument(sf2, "Accordion", new Patch(0, 21), organ);
326         newInstrument(sf2, "Accordion", new Patch(0, 22), organ);
327         newInstrument(sf2, "Accordion", new Patch(0, 23), organ);
328         newInstrument(sf2, "Guitar", new Patch(0, 24), guitar, guitar_pick);
329         newInstrument(sf2, "Guitar", new Patch(0, 25), guitar, guitar_pick);
330         newInstrument(sf2, "Guitar", new Patch(0, 26), guitar, guitar_pick);
331         newInstrument(sf2, "Guitar", new Patch(0, 27), guitar, guitar_pick);
332         newInstrument(sf2, "Guitar", new Patch(0, 28), guitar, guitar_pick);
333         newInstrument(sf2, "Distorted Guitar", new Patch(0, 29), guitar_dist);
334         newInstrument(sf2, "Distorted Guitar", new Patch(0, 30), guitar_dist);
335         newInstrument(sf2, "Guitar", new Patch(0, 31), guitar, guitar_pick);
336         newInstrument(sf2, "Finger Bass", new Patch(0, 32), bass1);
337         newInstrument(sf2, "Finger Bass", new Patch(0, 33), bass1);
338         newInstrument(sf2, "Finger Bass", new Patch(0, 34), bass1);
339         newInstrument(sf2, "Frettless Bass", new Patch(0, 35), bass2);
340         newInstrument(sf2, "Frettless Bass", new Patch(0, 36), bass2);
341         newInstrument(sf2, "Frettless Bass", new Patch(0, 37), bass2);
342         newInstrument(sf2, "Synth Bass1", new Patch(0, 38), synthbass);
343         newInstrument(sf2, "Synth Bass2", new Patch(0, 39), synthbass);
344         newInstrument(sf2, "Solo String", new Patch(0, 40), string2, solostring);
345         newInstrument(sf2, "Solo String", new Patch(0, 41), string2, solostring);
346         newInstrument(sf2, "Solo String", new Patch(0, 42), string2, solostring);
347         newInstrument(sf2, "Solo String", new Patch(0, 43), string2, solostring);
348         newInstrument(sf2, "Solo String", new Patch(0, 44), string2, solostring);
349         newInstrument(sf2, "Def", new Patch(0, 45), defaultsound);
350         newInstrument(sf2, "Harp", new Patch(0, 46), bell);
351         newInstrument(sf2, "Timpani", new Patch(0, 47), timpani);
352         newInstrument(sf2, "Strings", new Patch(0, 48), string2);
353         SF2Instrument slow_strings =
354                 newInstrument(sf2, "Slow Strings", new Patch(0, 49), string2);
355         SF2InstrumentRegion region = slow_strings.getRegions().get(0);
356         region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, 2500);
357         region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 2000);
358         newInstrument(sf2, "Synth Strings", new Patch(0, 50), string2);
359         newInstrument(sf2, "Synth Strings", new Patch(0, 51), string2);
360
361
362         newInstrument(sf2, "Choir", new Patch(0, 52), choir);
363         newInstrument(sf2, "Choir", new Patch(0, 53), choir);
364         newInstrument(sf2, "Choir", new Patch(0, 54), choir);
365         {
366             SF2Instrument ins = newInstrument(sf2, "Orch Hit",
367                     new Patch(0, 55), orchhit, orchhit, timpani);
368             region = ins.getRegions().get(0);
369             region.putInteger(SF2Region.GENERATOR_COARSETUNE, -12);
370             region.putInteger(SF2Region.GENERATOR_INITIALATTENUATION, -100);
```

```
        }
        newInstrument(sf2, "Trumpet", new Patch(0, 56), trumpet);
        newInstrument(sf2, "Trombone", new Patch(0, 57), trombone);
        newInstrument(sf2, "Trombone", new Patch(0, 58), trombone);
        newInstrument(sf2, "Trumpet", new Patch(0, 59), trumpet);
        newInstrument(sf2, "Horn", new Patch(0, 60), horn);
        newInstrument(sf2, "Brass␣Section", new Patch(0, 61), brass_section);
        newInstrument(sf2, "Brass␣Section", new Patch(0, 62), brass_section);
        newInstrument(sf2, "Brass␣Section", new Patch(0, 63), brass_section);
        newInstrument(sf2, "Sax", new Patch(0, 64), sax);
        newInstrument(sf2, "Sax", new Patch(0, 65), sax);
        newInstrument(sf2, "Sax", new Patch(0, 66), sax);
        newInstrument(sf2, "Sax", new Patch(0, 67), sax);
        newInstrument(sf2, "Oboe", new Patch(0, 68), oboe);
        newInstrument(sf2, "Horn", new Patch(0, 69), horn);
        newInstrument(sf2, "Bassoon", new Patch(0, 70), bassoon);
        newInstrument(sf2, "Clarinet", new Patch(0, 71), clarinet);
        newInstrument(sf2, "Flute", new Patch(0, 72), flute);
        newInstrument(sf2, "Flute", new Patch(0, 73), flute);
        newInstrument(sf2, "Flute", new Patch(0, 74), flute);
        newInstrument(sf2, "Flute", new Patch(0, 75), flute);
        newInstrument(sf2, "Flute", new Patch(0, 76), flute);
        newInstrument(sf2, "Flute", new Patch(0, 77), flute);
        newInstrument(sf2, "Flute", new Patch(0, 78), flute);
        newInstrument(sf2, "Flute", new Patch(0, 79), flute);
        newInstrument(sf2, "Organ", new Patch(0, 80), organ);
        newInstrument(sf2, "Organ", new Patch(0, 81), organ);
        newInstrument(sf2, "Flute", new Patch(0, 82), flute);
        newInstrument(sf2, "Organ", new Patch(0, 83), organ);
        newInstrument(sf2, "Organ", new Patch(0, 84), organ);
        newInstrument(sf2, "Choir", new Patch(0, 85), choir);
        newInstrument(sf2, "Organ", new Patch(0, 86), organ);
        newInstrument(sf2, "Organ", new Patch(0, 87), organ);
        newInstrument(sf2, "Synth␣Strings", new Patch(0, 88), string2);
        newInstrument(sf2, "Organ", new Patch(0, 89), organ);
        newInstrument(sf2, "Def", new Patch(0, 90), defaultsound);
        newInstrument(sf2, "Choir", new Patch(0, 91), choir);
        newInstrument(sf2, "Organ", new Patch(0, 92), organ);
        newInstrument(sf2, "Organ", new Patch(0, 93), organ);
        newInstrument(sf2, "Organ", new Patch(0, 94), organ);
        newInstrument(sf2, "Organ", new Patch(0, 95), organ);
        newInstrument(sf2, "Organ", new Patch(0, 96), organ);
        newInstrument(sf2, "Organ", new Patch(0, 97), organ);
        newInstrument(sf2, "Bell", new Patch(0, 98), bell);
        newInstrument(sf2, "Organ", new Patch(0, 99), organ);
        newInstrument(sf2, "Organ", new Patch(0, 100), organ);
        newInstrument(sf2, "Organ", new Patch(0, 101), organ);
        newInstrument(sf2, "Def", new Patch(0, 102), defaultsound);
        newInstrument(sf2, "Synth␣Strings", new Patch(0, 103), string2);
        newInstrument(sf2, "Def", new Patch(0, 104), defaultsound);
        newInstrument(sf2, "Def", new Patch(0, 105), defaultsound);
        newInstrument(sf2, "Def", new Patch(0, 106), defaultsound);
        newInstrument(sf2, "Def", new Patch(0, 107), defaultsound);
        newInstrument(sf2, "Marimba", new Patch(0, 108), bell);
        newInstrument(sf2, "Sax", new Patch(0, 109), sax);
        newInstrument(sf2, "Solo␣String", new Patch(0, 110), string2, solostring);
        newInstrument(sf2, "Oboe", new Patch(0, 111), oboe);
        newInstrument(sf2, "Bell", new Patch(0, 112), bell);
        newInstrument(sf2, "Melodic␣Toms", new Patch(0, 113), melodic_toms);
        newInstrument(sf2, "Marimba", new Patch(0, 114), bell);
        newInstrument(sf2, "Melodic␣Toms", new Patch(0, 115), melodic_toms);
        newInstrument(sf2, "Melodic␣Toms", new Patch(0, 116), melodic_toms);
```

```
        newInstrument(sf2, "Melodic Toms", new Patch(0, 117), melodic_toms);
        newInstrument(sf2, "Reverse Cymbal", new Patch(0, 118), reverse_cymbal);
        newInstrument(sf2, "Reverse Cymbal", new Patch(0, 119), reverse_cymbal);
        newInstrument(sf2, "Guitar", new Patch(0, 120), guitar);
        newInstrument(sf2, "Def", new Patch(0, 121), defaultsound);
        {
            SF2Instrument ins = newInstrument(sf2, "Seashore/Reverse Cymbal",
                    new Patch(0, 122), reverse_cymbal);
            region = ins.getRegions().get(0);
            region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
            region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 18500);
            region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 4500);
            region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, -4500);
        }
        {
            SF2Instrument ins = newInstrument(sf2, "Bird/Flute",
                    new Patch(0, 123), flute);
            region = ins.getRegions().get(0);
            region.putInteger(SF2Region.GENERATOR_COARSETUNE, 24);
            region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, -3000);
            region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
        }
        newInstrument(sf2, "Def", new Patch(0, 124), side_stick);
        {
            SF2Instrument ins = newInstrument(sf2, "Seashore/Reverse Cymbal",
                    new Patch(0, 125), reverse_cymbal);
            region = ins.getRegions().get(0);
            region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
            region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 18500);
            region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 4500);
            region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, -4500);
        }
        newInstrument(sf2, "Applause/crash_cymbal",
                new Patch(0, 126), crash_cymbal);
        newInstrument(sf2, "Gunshot/side_stick", new Patch(0, 127), side_stick);

        for (SF2Instrument instrument : sf2.getInstruments()) {
            Patch patch = instrument.getPatch();
            if (patch instanceof ModelPatch) {
                if (((ModelPatch) patch).isPercussion())
                    continue;
            }
            instrument.setName(general_midi_instruments[patch.getProgram()]);
        }

        return sf2;

    }

    public static SF2Layer new_bell(SF2Soundbank sf2) {
        Random random = new Random(102030201);
        int x = 8;
        int fftsize = 4096 * x;
        double[] data = new double[fftsize * 2];
        double base = x * 25;
        double start_w = 0.01;
        double end_w = 0.05;
        double start_a = 0.2;
        double end_a = 0.00001;
        double a = start_a;
        double a_step = Math.pow(end_a / start_a, 1.0 / 40.0);
        for (int i = 0; i < 40; i++) {
```

```
            double detune = 1 + (random.nextDouble() * 2 - 1) * 0.01;
            double w = start_w + (end_w - start_w) * (i / 40.0);
            complexGaussianDist(data, base * (i + 1) * detune, w, a);
            a *= a_step;
        }
        SF2Sample sample = newSimpleFFTSample(sf2, "EPiano", data, base);
        SF2Layer layer = newLayer(sf2, "EPiano", sample);
        SF2Region region = layer.getRegions().get(0);
        region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
        region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -12000);
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 0);
        region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
        region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
        region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, 1200);
        region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
        region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, -9000);
        region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 16000);
        return layer;
    }

    public static SF2Layer new_guitar1(SF2Soundbank sf2) {

        int x = 8;
        int fftsize = 4096 * x;
        double[] data = new double[fftsize * 2];
        double base = x * 25;
        double start_w = 0.01;
        double end_w = 0.01;
        double start_a = 2;
        double end_a = 0.01;
        double a = start_a;
        double a_step = Math.pow(end_a / start_a, 1.0 / 40.0);

        double[] aa = new double[40];
        for (int i = 0; i < 40; i++) {
            aa[i] = a;
            a *= a_step;
        }

        aa[0] = 2;
        aa[1] = 0.5;
        aa[2] = 0.45;
        aa[3] = 0.2;
        aa[4] = 1;
        aa[5] = 0.5;
        aa[6] = 2;
        aa[7] = 1;
        aa[8] = 0.5;
        aa[9] = 1;
        aa[9] = 0.5;
        aa[10] = 0.2;
        aa[11] = 1;
        aa[12] = 0.7;
        aa[13] = 0.5;
        aa[14] = 1;

        for (int i = 0; i < 40; i++) {
            double w = start_w + (end_w - start_w) * (i / 40.0);
            complexGaussianDist(data, base * (i + 1), w, aa[i]);
        }

        SF2Sample sample = newSimpleFFTSample(sf2, "Guitar", data, base);
```

```
        SF2Layer layer = newLayer(sf2, "Guitar", sample);
        SF2Region region = layer.getRegions().get(0);
        region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
        region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -12000);
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 0);
        region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 2400);
        region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);

        region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, -100);
        region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
        region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, -6000);
        region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 16000);
        region.putInteger(SF2Region.GENERATOR_INITIALATTENUATION, -20);
        return layer;
    }

    public static SF2Layer new_guitar_dist(SF2Soundbank sf2) {

        int x = 8;
        int fftsize = 4096 * x;
        double[] data = new double[fftsize * 2];
        double base = x * 25;
        double start_w = 0.01;
        double end_w = 0.01;
        double start_a = 2;
        double end_a = 0.01;
        double a = start_a;
        double a_step = Math.pow(end_a / start_a, 1.0 / 40.0);

        double[] aa = new double[40];
        for (int i = 0; i < 40; i++) {
            aa[i] = a;
            a *= a_step;
        }

        aa[0] = 5;
        aa[1] = 2;
        aa[2] = 0.45;
        aa[3] = 0.2;
        aa[4] = 1;
        aa[5] = 0.5;
        aa[6] = 2;
        aa[7] = 1;
        aa[8] = 0.5;
        aa[9] = 1;
        aa[9] = 0.5;
        aa[10] = 0.2;
        aa[11] = 1;
        aa[12] = 0.7;
        aa[13] = 0.5;
        aa[14] = 1;

        for (int i = 0; i < 40; i++) {
            double w = start_w + (end_w - start_w) * (i / 40.0);
            complexGaussianDist(data, base * (i + 1), w, aa[i]);
        }


        SF2Sample sample = newSimpleFFTSample_dist(sf2, "Distorted Guitar",
                data, base, 10000.0);


```

```
        SF2Layer layer = newLayer(sf2, "Distorted Guitar", sample);
        SF2Region region = layer.getRegions().get(0);
        region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
        region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -12000);
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 0);
        //region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 2400);
        //region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 200);

        //region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, -100);
        //region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
        //region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, -1000);
        region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 8000);
        //region.putInteger(SF2Region.GENERATOR_INITIALATTENUATION, -20);
        return layer;
    }

    public static SF2Layer new_guitar_pick(SF2Soundbank sf2) {

        double datab[];

        // Make treble part
        {
            int m = 2;
            int fftlen = 4096 * m;
            double[] data = new double[2 * fftlen];
            Random random = new Random(3049912);
            for (int i = 0; i < data.length; i += 2)
                data[i] = (2.0 * (random.nextDouble() - 0.5));
            fft(data);
            // Remove all negative frequency
            for (int i = fftlen / 2; i < data.length; i++)
                data[i] = 0;
            for (int i = 0; i < 2048 * m; i++) {
                data[i] *= Math.exp(-Math.abs((i - 23) / ((double) m)) * 1.2)
                        + Math.exp(-Math.abs((i - 40) / ((double) m)) * 0.9);
            }
            randomPhase(data, new Random(3049912));
            ifft(data);
            normalize(data, 0.8);
            data = realPart(data);
            double gain = 1.0;
            for (int i = 0; i < data.length; i++) {
                data[i] *= gain;
                gain *= 0.9994;
            }
            datab = data;

            fadeUp(data, 80);
        }

        SF2Sample sample = newSimpleDrumSample(sf2, "Guitar Noise", datab);

        SF2Layer layer = new SF2Layer(sf2);
        layer.setName("Guitar Noise");

        SF2GlobalRegion global = new SF2GlobalRegion();
        layer.setGlobalZone(global);
        sf2.addResource(layer);

        SF2LayerRegion region = new SF2LayerRegion();
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 12000);
        //region.putInteger(SF2Region.GENERATOR_SCALETUNING, 0);
```

```
//     region.putInteger(SF2Region.GENERATOR_INITIALATTENUATION, -100);
/*
        region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, 0);
        region.putInteger(SF2Region.GENERATOR_SUSTAINMODENV, 1000);
        region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
        region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, -11000);
        region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 12000);
         */

        region.setSample(sample);
        layer.getRegions().add(region);

        return layer;
    }

    public static SF2Layer new_gpiano(SF2Soundbank sf2) {
        //Random random = new Random(302030201);
        int x = 8;
        int fftsize = 4096 * x;
        double[] data = new double[fftsize * 2];
        double base = x * 25;
        double start_a = 0.2;
        double end_a = 0.001;
        double a = start_a;
        double a_step = Math.pow(end_a / start_a, 1.0 / 15.0);

        double[] aa = new double[30];
        for (int i = 0; i < 30; i++) {
            aa[i] = a;
            a *= a_step;
        }

        aa[0] *= 2;
        //aa[2] *= 0.1;
        aa[4] *= 2;


        aa[12] *= 0.9;
        aa[13] *= 0.7;
        for (int i = 14; i < 30; i++) {
            aa[i] *= 0.5;
        }


        for (int i = 0; i < 30; i++) {
            //double detune = 1 + (random.nextDouble()*2 - 1)*0.0001;
            double w = 0.2;
            double ai = aa[i];
            if (i > 10) {
                w = 5;
                ai *= 10;
            }
            int adjust = 0;
            if (i > 5) {
                adjust = (i - 5) * 7;
            }
            complexGaussianDist(data, base * (i + 1) + adjust, w, ai);
        }

        SF2Sample sample = newSimpleFFTSample(sf2, "Grand Piano", data, base, 200);
        SF2Layer layer = newLayer(sf2, "Grand Piano", sample);
        SF2Region region = layer.getRegions().get(0);
```

```
        region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
        region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -7000);
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 0);
        region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
        region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
        region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, -6000);
        region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
        region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, -5500);
        region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 18000);
        return layer;
    }

    public static SF2Layer new_gpiano2(SF2Soundbank sf2) {
        //Random random = new Random(302030201);
        int x = 8;
        int fftsize = 4096 * x;
        double[] data = new double[fftsize * 2];
        double base = x * 25;
        double start_a = 0.2;
        double end_a = 0.001;
        double a = start_a;
        double a_step = Math.pow(end_a / start_a, 1.0 / 20.0);

        double[] aa = new double[30];
        for (int i = 0; i < 30; i++) {
            aa[i] = a;
            a *= a_step;
        }

        aa[0] *= 1;
        //aa[2] *= 0.1;
        aa[4] *= 2;


        aa[12] *= 0.9;
        aa[13] *= 0.7;
        for (int i = 14; i < 30; i++) {
            aa[i] *= 0.5;
        }


        for (int i = 0; i < 30; i++) {
            //double detune = 1 + (random.nextDouble()*2 - 1)*0.0001;
            double w = 0.2;
            double ai = aa[i];
            if (i > 10) {
                w = 5;
                ai *= 10;
            }
            int adjust = 0;
            if (i > 5) {
                adjust = (i - 5) * 7;
            }
            complexGaussianDist(data, base * (i + 1) + adjust, w, ai);
        }

        SF2Sample sample = newSimpleFFTSample(sf2, "Grand Piano", data, base, 200);
        SF2Layer layer = newLayer(sf2, "Grand Piano", sample);
        SF2Region region = layer.getRegions().get(0);
        region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
        region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -7000);
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 0);
```

```
805         region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
806         region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
807         region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, -6000);
808         region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
809         region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, -5500);
810         region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 18000);
811         return layer;
812     }
813 
814     public static SF2Layer new_piano_hammer(SF2Soundbank sf2) {
815 
816         double datab[];
817 
818         // Make treble part
819         {
820             int m = 2;
821             int fftlen = 4096 * m;
822             double[] data = new double[2 * fftlen];
823             Random random = new Random(3049912);
824             for (int i = 0; i < data.length; i += 2)
825                 data[i] = (2.0 * (random.nextDouble() - 0.5));
826             fft(data);
827             // Remove all negative frequency
828             for (int i = fftlen / 2; i < data.length; i++)
829                 data[i] = 0;
830             for (int i = 0; i < 2048 * m; i++)
831                 data[i] *= Math.exp(-Math.abs((i - 37) / ((double) m)) * 0.05);
832             randomPhase(data, new Random(3049912));
833             ifft(data);
834             normalize(data, 0.6);
835             data = realPart(data);
836             double gain = 1.0;
837             for (int i = 0; i < data.length; i++) {
838                 data[i] *= gain;
839                 gain *= 0.9997;
840             }
841             datab = data;
842 
843             fadeUp(data, 80);
844         }
845 
846         SF2Sample sample = newSimpleDrumSample(sf2, "Piano Hammer", datab);
847 
848         SF2Layer layer = new SF2Layer(sf2);
849         layer.setName("Piano Hammer");
850 
851         SF2GlobalRegion global = new SF2GlobalRegion();
852         layer.setGlobalZone(global);
853         sf2.addResource(layer);
854 
855         SF2LayerRegion region = new SF2LayerRegion();
856         region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 12000);
857         //region.putInteger(SF2Region.GENERATOR_SCALETUNING, 0);
858 /*
859         region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, 0);
860         region.putInteger(SF2Region.GENERATOR_SUSTAINMODENV, 1000);
861         region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
862         region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, -11000);
863         region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 12000);
864          */
865 
866         region.setSample(sample);
```

```
867         layer.getRegions().add(region);
868 
869         return layer;
870     }
871 
872     public static SF2Layer new_piano1(SF2Soundbank sf2) {
873         //Random random = new Random(302030201);
874         int x = 8;
875         int fftsize = 4096 * x;
876         double[] data = new double[fftsize * 2];
877         double base = x * 25;
878         double start_a = 0.2;
879         double end_a = 0.0001;
880         double a = start_a;
881         double a_step = Math.pow(end_a / start_a, 1.0 / 40.0);
882 
883         double[] aa = new double[30];
884         for (int i = 0; i < 30; i++) {
885             aa[i] = a;
886             a *= a_step;
887         }
888 
889         aa[0] *= 5;
890         aa[2] *= 0.1;
891         aa[7] *= 5;
892 
893 
894         for (int i = 0; i < 30; i++) {
895             //double detune = 1 + (random.nextDouble()*2 - 1)*0.0001;
896             double w = 0.2;
897             double ai = aa[i];
898             if (i > 12) {
899                 w = 5;
900                 ai *= 10;
901             }
902             int adjust = 0;
903             if (i > 5) {
904                 adjust = (i - 5) * 7;
905             }
906             complexGaussianDist(data, base * (i + 1) + adjust, w, ai);
907         }
908 
909         complexGaussianDist(data, base * (15.5), 1, 0.1);
910         complexGaussianDist(data, base * (17.5), 1, 0.01);
911 
912         SF2Sample sample = newSimpleFFTSample(sf2, "EPiano", data, base, 200);
913         SF2Layer layer = newLayer(sf2, "EPiano", sample);
914         SF2Region region = layer.getRegions().get(0);
915         region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
916         region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -12000);
917         region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 0);
918         region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
919         region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
920         region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, -1200);
921         region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
922         region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, -5500);
923         region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 16000);
924         return layer;
925     }
926 
927     public static SF2Layer new_epiano1(SF2Soundbank sf2) {
928         Random random = new Random(302030201);
```

```
        int x = 8;
        int fftsize = 4096 * x;
        double[] data = new double[fftsize * 2];
        double base = x * 25;
        double start_w = 0.05;
        double end_w = 0.05;
        double start_a = 0.2;
        double end_a = 0.0001;
        double a = start_a;
        double a_step = Math.pow(end_a / start_a, 1.0 / 40.0);
        for (int i = 0; i < 40; i++) {
            double detune = 1 + (random.nextDouble() * 2 - 1) * 0.0001;
            double w = start_w + (end_w - start_w) * (i / 40.0);
            complexGaussianDist(data, base * (i + 1) * detune, w, a);
            a *= a_step;
        }



        SF2Sample sample = newSimpleFFTSample(sf2, "EPiano", data, base);
        SF2Layer layer = newLayer(sf2, "EPiano", sample);
        SF2Region region = layer.getRegions().get(0);
        region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
        region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -12000);
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 0);
        region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
        region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
        region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, 1200);
        region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
        region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, -9000);
        region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 16000);
        return layer;
    }

    public static SF2Layer new_epiano2(SF2Soundbank sf2) {
        Random random = new Random(302030201);
        int x = 8;
        int fftsize = 4096 * x;
        double[] data = new double[fftsize * 2];
        double base = x * 25;
        double start_w = 0.01;
        double end_w = 0.05;
        double start_a = 0.2;
        double end_a = 0.00001;
        double a = start_a;
        double a_step = Math.pow(end_a / start_a, 1.0 / 40.0);
        for (int i = 0; i < 40; i++) {
            double detune = 1 + (random.nextDouble() * 2 - 1) * 0.0001;
            double w = start_w + (end_w - start_w) * (i / 40.0);
            complexGaussianDist(data, base * (i + 1) * detune, w, a);
            a *= a_step;
        }

        SF2Sample sample = newSimpleFFTSample(sf2, "EPiano", data, base);
        SF2Layer layer = newLayer(sf2, "EPiano", sample);
        SF2Region region = layer.getRegions().get(0);
        region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
        region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -12000);
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 0);
        region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 8000);
        region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
        region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, 2400);
```

```
        region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
        region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, -9000);
        region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 16000);
        region.putInteger(SF2Region.GENERATOR_INITIALATTENUATION, -100);
        return layer;
    }

    public static SF2Layer new_bass1(SF2Soundbank sf2) {
        int x = 8;
        int fftsize = 4096 * x;
        double[] data = new double[fftsize * 2];
        double base = x * 25;
        double start_w = 0.05;
        double end_w = 0.05;
        double start_a = 0.2;
        double end_a = 0.02;
        double a = start_a;
        double a_step = Math.pow(end_a / start_a, 1.0 / 25.0);

        double[] aa = new double[25];
        for (int i = 0; i < 25; i++) {
            aa[i] = a;
            a *= a_step;
        }

        aa[0] *= 8;
        aa[1] *= 4;
        aa[3] *= 8;
        aa[5] *= 8;

        for (int i = 0; i < 25; i++) {
            double w = start_w + (end_w - start_w) * (i / 40.0);
            complexGaussianDist(data, base * (i + 1), w, aa[i]);
        }


        SF2Sample sample = newSimpleFFTSample(sf2, "Bass", data, base);
        SF2Layer layer = newLayer(sf2, "Bass", sample);
        SF2Region region = layer.getRegions().get(0);
        region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
        region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -12000);
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 0);
        region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
        region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
        region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, -3000);
        region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
        region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, -5000);
        region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 11000);
        region.putInteger(SF2Region.GENERATOR_INITIALATTENUATION, -100);
        return layer;
    }

    public static SF2Layer new_synthbass(SF2Soundbank sf2) {
        int x = 8;
        int fftsize = 4096 * x;
        double[] data = new double[fftsize * 2];
        double base = x * 25;
        double start_w = 0.05;
        double end_w = 0.05;
        double start_a = 0.2;
        double end_a = 0.02;
        double a = start_a;
```

```
        double a_step = Math.pow(end_a / start_a, 1.0 / 25.0);

        double[] aa = new double[25];
        for (int i = 0; i < 25; i++) {
            aa[i] = a;
            a *= a_step;
        }

        aa[0] *= 16;
        aa[1] *= 4;
        aa[3] *= 16;
        aa[5] *= 8;

        for (int i = 0; i < 25; i++) {
            double w = start_w + (end_w - start_w) * (i / 40.0);
            complexGaussianDist(data, base * (i + 1), w, aa[i]);
        }


        SF2Sample sample = newSimpleFFTSample(sf2, "Bass", data, base);
        SF2Layer layer = newLayer(sf2, "Bass", sample);
        SF2Region region = layer.getRegions().get(0);
        region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
        region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -12000);
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 0);
        region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
        region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
        region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, -3000);
        region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
        region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, -3000);
        region.putInteger(SF2Region.GENERATOR_INITIALFILTERQ, 100);
        region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 8000);
        region.putInteger(SF2Region.GENERATOR_INITIALATTENUATION, -100);
        return layer;
    }

    public static SF2Layer new_bass2(SF2Soundbank sf2) {
        int x = 8;
        int fftsize = 4096 * x;
        double[] data = new double[fftsize * 2];
        double base = x * 25;
        double start_w = 0.05;
        double end_w = 0.05;
        double start_a = 0.2;
        double end_a = 0.002;
        double a = start_a;
        double a_step = Math.pow(end_a / start_a, 1.0 / 25.0);

        double[] aa = new double[25];
        for (int i = 0; i < 25; i++) {
            aa[i] = a;
            a *= a_step;
        }

        aa[0] *= 8;
        aa[1] *= 4;
        aa[3] *= 8;
        aa[5] *= 8;

        for (int i = 0; i < 25; i++) {
            double w = start_w + (end_w - start_w) * (i / 40.0);
            complexGaussianDist(data, base * (i + 1), w, aa[i]);
```

```
        }


        SF2Sample sample = newSimpleFFTSample(sf2, "Bass2", data, base);
        SF2Layer layer = newLayer(sf2, "Bass2", sample);
        SF2Region region = layer.getRegions().get(0);
        region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
        region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -8000);
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 0);
        region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
        region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
        region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, -6000);
        region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
        region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 5000);
        region.putInteger(SF2Region.GENERATOR_INITIALATTENUATION, -100);
        return layer;
    }

    public static SF2Layer new_solostring(SF2Soundbank sf2) {
        int x = 8;
        int fftsize = 4096 * x;
        double[] data = new double[fftsize * 2];
        double base = x * 25;
        double start_w = 2;
        double end_w = 2;
        double start_a = 0.2;
        double end_a = 0.01;

        double[] aa = new double[18];
        double a = start_a;
        double a_step = Math.pow(end_a / start_a, 1.0 / 40.0);
        for (int i = 0; i < aa.length; i++) {
            a *= a_step;
            aa[i] = a;
        }

        aa[0] *= 5;
        aa[1] *= 5;
        aa[2] *= 5;
        aa[3] *= 4;
        aa[4] *= 4;
        aa[5] *= 3;
        aa[6] *= 3;
        aa[7] *= 2;

        for (int i = 0; i < aa.length; i++) {
            double w = start_w + (end_w - start_w) * (i / 40.0);
            complexGaussianDist(data, base * (i + 1), w, a);
        }
        SF2Sample sample = newSimpleFFTSample(sf2, "Strings", data, base);
        SF2Layer layer = newLayer(sf2, "Strings", sample);
        SF2Region region = layer.getRegions().get(0);
        region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
        region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -5000);
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 1000);
        region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
        region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, -100);
        region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 9500);
        region.putInteger(SF2Region.GENERATOR_FREQVIBLFO, -1000);
        region.putInteger(SF2Region.GENERATOR_VIBLFOTOPITCH, 15);
        return layer;

```

```
    }

    public static SF2Layer new_orchhit(SF2Soundbank sf2) {
        int x = 8;
        int fftsize = 4096 * x;
        double[] data = new double[fftsize * 2];
        double base = x * 25;
        double start_w = 2;
        double end_w = 80;
        double start_a = 0.2;
        double end_a = 0.001;
        double a = start_a;
        double a_step = Math.pow(end_a / start_a, 1.0 / 40.0);
        for (int i = 0; i < 40; i++) {
            double w = start_w + (end_w - start_w) * (i / 40.0);
            complexGaussianDist(data, base * (i + 1), w, a);
            a *= a_step;
        }
        complexGaussianDist(data, base * 4, 300, 1);


        SF2Sample sample = newSimpleFFTSample(sf2, "Och Strings", data, base);
        SF2Layer layer = newLayer(sf2, "Och Strings", sample);
        SF2Region region = layer.getRegions().get(0);
        region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
        region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -5000);
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 200);
        region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 200);
        region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
        region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 9500);
        return layer;

    }

    public static SF2Layer new_string2(SF2Soundbank sf2) {
        int x = 8;
        int fftsize = 4096 * x;
        double[] data = new double[fftsize * 2];
        double base = x * 25;
        double start_w = 2;
        double end_w = 80;
        double start_a = 0.2;
        double end_a = 0.001;
        double a = start_a;
        double a_step = Math.pow(end_a / start_a, 1.0 / 40.0);
        for (int i = 0; i < 40; i++) {
            double w = start_w + (end_w - start_w) * (i / 40.0);
            complexGaussianDist(data, base * (i + 1), w, a);
            a *= a_step;
        }
        SF2Sample sample = newSimpleFFTSample(sf2, "Strings", data, base);
        SF2Layer layer = newLayer(sf2, "Strings", sample);
        SF2Region region = layer.getRegions().get(0);
        region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
        region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -5000);
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 1000);
        region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
        region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, -100);
        region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 9500);
        return layer;

    }
```

```
1239
1240    public static SF2Layer new_choir(SF2Soundbank sf2) {
1241        int x = 8;
1242        int fftsize = 4096 * x;
1243        double[] data = new double[fftsize * 2];
1244        double base = x * 25;
1245        double start_w = 2;
1246        double end_w = 80;
1247        double start_a = 0.2;
1248        double end_a = 0.001;
1249        double a = start_a;
1250        double a_step = Math.pow(end_a / start_a, 1.0 / 40.0);
1251        double[] aa = new double[40];
1252        for (int i = 0; i < aa.length; i++) {
1253            a *= a_step;
1254            aa[i] = a;
1255        }
1256
1257        aa[5] *= 0.1;
1258        aa[6] *= 0.01;
1259        aa[7] *= 0.1;
1260        aa[8] *= 0.1;
1261
1262        for (int i = 0; i < aa.length; i++) {
1263            double w = start_w + (end_w - start_w) * (i / 40.0);
1264            complexGaussianDist(data, base * (i + 1), w, aa[i]);
1265        }
1266        SF2Sample sample = newSimpleFFTSample(sf2, "Strings", data, base);
1267        SF2Layer layer = newLayer(sf2, "Strings", sample);
1268        SF2Region region = layer.getRegions().get(0);
1269        region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
1270        region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -5000);
1271        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 1000);
1272        region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
1273        region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, -100);
1274        region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 9500);
1275        return layer;
1276
1277    }
1278
1279    public static SF2Layer new_organ(SF2Soundbank sf2) {
1280        Random random = new Random(102030201);
1281        int x = 1;
1282        int fftsize = 4096 * x;
1283        double[] data = new double[fftsize * 2];
1284        double base = x * 15;
1285        double start_w = 0.01;
1286        double end_w = 0.01;
1287        double start_a = 0.2;
1288        double end_a = 0.001;
1289        double a = start_a;
1290        double a_step = Math.pow(end_a / start_a, 1.0 / 40.0);
1291
1292        for (int i = 0; i < 12; i++) {
1293            double w = start_w + (end_w - start_w) * (i / 40.0);
1294            complexGaussianDist(data, base * (i + 1), w,
1295                    a * (0.5 + 3 * (random.nextDouble())));
1296            a *= a_step;
1297        }
1298        SF2Sample sample = newSimpleFFTSample(sf2, "Organ", data, base);
1299        SF2Layer layer = newLayer(sf2, "Organ", sample);
1300        SF2Region region = layer.getRegions().get(0);
```

```
        region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
        region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -6000);
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, -1000);
        region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
        region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, -100);
        region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 9500);
        return layer;

    }

    public static SF2Layer new_ch_organ(SF2Soundbank sf2) {
        int x = 1;
        int fftsize = 4096 * x;
        double[] data = new double[fftsize * 2];
        double base = x * 15;
        double start_w = 0.01;
        double end_w = 0.01;
        double start_a = 0.2;
        double end_a = 0.001;
        double a = start_a;
        double a_step = Math.pow(end_a / start_a, 1.0 / 60.0);

        double[] aa = new double[60];
        for (int i = 0; i < aa.length; i++) {
            a *= a_step;
            aa[i] = a;
        }

        aa[0] *= 5;
        aa[1] *= 2;
        aa[2] = 0;
        aa[4] = 0;
        aa[5] = 0;
        aa[7] *= 7;
        aa[9] = 0;
        aa[10] = 0;
        aa[12] = 0;
        aa[15] *= 7;
        aa[18] = 0;
        aa[20] = 0;
        aa[24] = 0;
        aa[27] *= 5;
        aa[29] = 0;
        aa[30] = 0;
        aa[33] = 0;
        aa[36] *= 4;
        aa[37] = 0;
        aa[39] = 0;
        aa[42] = 0;
        aa[43] = 0;
        aa[47] = 0;
        aa[50] *= 4;
        aa[52] = 0;
        aa[55] = 0;
        aa[57] = 0;


        aa[10] *= 0.1;
        aa[11] *= 0.1;
        aa[12] *= 0.1;
        aa[13] *= 0.1;

```

```
        aa[17] *= 0.1;
        aa[18] *= 0.1;
        aa[19] *= 0.1;
        aa[20] *= 0.1;

        for (int i = 0; i < 60; i++) {
            double w = start_w + (end_w - start_w) * (i / 40.0);
            complexGaussianDist(data, base * (i + 1), w, aa[i]);
            a *= a_step;
        }
        SF2Sample sample = newSimpleFFTSample(sf2, "Organ", data, base);
        SF2Layer layer = newLayer(sf2, "Organ", sample);
        SF2Region region = layer.getRegions().get(0);
        region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
        region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -10000);
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, -1000);
        return layer;

    }

    public static SF2Layer new_flute(SF2Soundbank sf2) {
        int x = 8;
        int fftsize = 4096 * x;
        double[] data = new double[fftsize * 2];
        double base = x * 15;

        complexGaussianDist(data, base * 1, 0.001, 0.5);
        complexGaussianDist(data, base * 2, 0.001, 0.5);
        complexGaussianDist(data, base * 3, 0.001, 0.5);
        complexGaussianDist(data, base * 4, 0.01, 0.5);

        complexGaussianDist(data, base * 4, 100, 120);
        complexGaussianDist(data, base * 6, 100, 40);
        complexGaussianDist(data, base * 8, 100, 80);

        complexGaussianDist(data, base * 5, 0.001, 0.05);
        complexGaussianDist(data, base * 6, 0.001, 0.06);
        complexGaussianDist(data, base * 7, 0.001, 0.04);
        complexGaussianDist(data, base * 8, 0.005, 0.06);
        complexGaussianDist(data, base * 9, 0.005, 0.06);
        complexGaussianDist(data, base * 10, 0.01, 0.1);
        complexGaussianDist(data, base * 11, 0.08, 0.7);
        complexGaussianDist(data, base * 12, 0.08, 0.6);
        complexGaussianDist(data, base * 13, 0.08, 0.6);
        complexGaussianDist(data, base * 14, 0.08, 0.6);
        complexGaussianDist(data, base * 15, 0.08, 0.5);
        complexGaussianDist(data, base * 16, 0.08, 0.5);
        complexGaussianDist(data, base * 17, 0.08, 0.2);


        complexGaussianDist(data, base * 1, 10, 8);
        complexGaussianDist(data, base * 2, 10, 8);
        complexGaussianDist(data, base * 3, 10, 8);
        complexGaussianDist(data, base * 4, 10, 8);
        complexGaussianDist(data, base * 5, 10, 8);
        complexGaussianDist(data, base * 6, 20, 9);
        complexGaussianDist(data, base * 7, 20, 9);
        complexGaussianDist(data, base * 8, 20, 9);
        complexGaussianDist(data, base * 9, 20, 8);
        complexGaussianDist(data, base * 10, 30, 8);
        complexGaussianDist(data, base * 11, 30, 9);
        complexGaussianDist(data, base * 12, 30, 9);
```

```
        complexGaussianDist(data, base * 13, 30, 8);
        complexGaussianDist(data, base * 14, 30, 8);
        complexGaussianDist(data, base * 15, 30, 7);
        complexGaussianDist(data, base * 16, 30, 7);
        complexGaussianDist(data, base * 17, 30, 6);

        SF2Sample sample = newSimpleFFTSample(sf2, "Flute", data, base);
        SF2Layer layer = newLayer(sf2, "Flute", sample);
        SF2Region region = layer.getRegions().get(0);
        region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
        region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -6000);
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, -1000);
        region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
        region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, -100);
        region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 9500);
        return layer;

    }

    public static SF2Layer new_horn(SF2Soundbank sf2) {
        int x = 8;
        int fftsize = 4096 * x;
        double[] data = new double[fftsize * 2];
        double base = x * 15;

        double start_a = 0.5;
        double end_a = 0.00000000001;
        double a = start_a;
        double a_step = Math.pow(end_a / start_a, 1.0 / 40.0);
        for (int i = 0; i < 40; i++) {
            if (i == 0)
                complexGaussianDist(data, base * (i + 1), 0.1, a * 0.2);
            else
                complexGaussianDist(data, base * (i + 1), 0.1, a);
            a *= a_step;
        }

        complexGaussianDist(data, base * 2, 100, 1);


        SF2Sample sample = newSimpleFFTSample(sf2, "Horn", data, base);
        SF2Layer layer = newLayer(sf2, "Horn", sample);
        SF2Region region = layer.getRegions().get(0);
        region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
        region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -6000);
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, -1000);
        region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
        region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, -100);

        region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, -500);
        region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
        region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, 5000);
        region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 4500);
        return layer;

    }

    public static SF2Layer new_trumpet(SF2Soundbank sf2) {
        int x = 8;
        int fftsize = 4096 * x;
        double[] data = new double[fftsize * 2];
        double base = x * 15;
```

```
        double start_a = 0.5;
        double end_a = 0.00001;
        double a = start_a;
        double a_step = Math.pow(end_a / start_a, 1.0 / 80.0);
        double[] aa = new double[80];
        for (int i = 0; i < 80; i++) {
            aa[i] = a;
            a *= a_step;
        }

        aa[0] *= 0.05;
        aa[1] *= 0.2;
        aa[2] *= 0.5;
        aa[3] *= 0.85;

        for (int i = 0; i < 80; i++) {
            complexGaussianDist(data, base * (i + 1), 0.1, aa[i]);
        }

        complexGaussianDist(data, base * 5, 300, 3);


        SF2Sample sample = newSimpleFFTSample(sf2, "Trumpet", data, base);
        SF2Layer layer = newLayer(sf2, "Trumpet", sample);
        SF2Region region = layer.getRegions().get(0);
        region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
        region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -10000);
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 0);
        region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
        region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, -100);

        region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, -4000);
        region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, -2500);
        region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, 5000);
        region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 4500);
        region.putInteger(SF2Region.GENERATOR_INITIALFILTERQ, 10);
        return layer;

    }

    public static SF2Layer new_brass_section(SF2Soundbank sf2) {
        int x = 8;
        int fftsize = 4096 * x;
        double[] data = new double[fftsize * 2];
        double base = x * 15;

        double start_a = 0.5;
        double end_a = 0.005;
        double a = start_a;
        double a_step = Math.pow(end_a / start_a, 1.0 / 30.0);
        double[] aa = new double[30];
        for (int i = 0; i < 30; i++) {
            aa[i] = a;
            a *= a_step;
        }

        aa[0] *= 0.8;
        aa[1] *= 0.9;

        double w = 5;
        for (int i = 0; i < 30; i++) {
```

```
            complexGaussianDist(data, base * (i + 1), 0.1 * w, aa[i] * w);
            w += 6; //*= w_step;
        }

        complexGaussianDist(data, base * 6, 300, 2);


        SF2Sample sample = newSimpleFFTSample(sf2, "Brass Section", data, base);
        SF2Layer layer = newLayer(sf2, "Brass Section", sample);
        SF2Region region = layer.getRegions().get(0);
        region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
        region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -9200);
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, -1000);
        region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
        region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, -100);

        region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, -3000);
        region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
        region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, 5000);
        region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 4500);
        return layer;

    }

    public static SF2Layer new_trombone(SF2Soundbank sf2) {
        int x = 8;
        int fftsize = 4096 * x;
        double[] data = new double[fftsize * 2];
        double base = x * 15;

        double start_a = 0.5;
        double end_a = 0.001;
        double a = start_a;
        double a_step = Math.pow(end_a / start_a, 1.0 / 80.0);
        double[] aa = new double[80];
        for (int i = 0; i < 80; i++) {
            aa[i] = a;
            a *= a_step;
        }

        aa[0] *= 0.3;
        aa[1] *= 0.7;

        for (int i = 0; i < 80; i++) {
            complexGaussianDist(data, base * (i + 1), 0.1, aa[i]);
        }

        complexGaussianDist(data, base * 6, 300, 2);


        SF2Sample sample = newSimpleFFTSample(sf2, "Trombone", data, base);
        SF2Layer layer = newLayer(sf2, "Trombone", sample);
        SF2Region region = layer.getRegions().get(0);
        region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
        region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -8000);
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, -1000);
        region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
        region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, -100);

        region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, -2000);
        region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
        region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, 5000);
```

```
        region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 4500);
        region.putInteger(SF2Region.GENERATOR_INITIALFILTERQ, 10);
        return layer;

    }

    public static SF2Layer new_sax(SF2Soundbank sf2) {
        int x = 8;
        int fftsize = 4096 * x;
        double[] data = new double[fftsize * 2];
        double base = x * 15;

        double start_a = 0.5;
        double end_a = 0.01;
        double a = start_a;
        double a_step = Math.pow(end_a / start_a, 1.0 / 40.0);
        for (int i = 0; i < 40; i++) {
            if (i == 0 || i == 2)
                complexGaussianDist(data, base * (i + 1), 0.1, a * 4);
            else
                complexGaussianDist(data, base * (i + 1), 0.1, a);
            a *= a_step;
        }

        complexGaussianDist(data, base * 4, 200, 1);

        SF2Sample sample = newSimpleFFTSample(sf2, "Sax", data, base);
        SF2Layer layer = newLayer(sf2, "Sax", sample);
        SF2Region region = layer.getRegions().get(0);
        region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
        region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -6000);
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, -1000);
        region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
        region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, -100);

        region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, -3000);
        region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
        region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, 5000);
        region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 4500);
        return layer;

    }

    public static SF2Layer new_oboe(SF2Soundbank sf2) {
        int x = 8;
        int fftsize = 4096 * x;
        double[] data = new double[fftsize * 2];
        double base = x * 15;

        complexGaussianDist(data, base * 5, 100, 80);


        complexGaussianDist(data, base * 1, 0.01, 0.53);
        complexGaussianDist(data, base * 2, 0.01, 0.51);
        complexGaussianDist(data, base * 3, 0.01, 0.48);
        complexGaussianDist(data, base * 4, 0.01, 0.49);
        complexGaussianDist(data, base * 5, 0.01, 5);
        complexGaussianDist(data, base * 6, 0.01, 0.51);
        complexGaussianDist(data, base * 7, 0.01, 0.50);
        complexGaussianDist(data, base * 8, 0.01, 0.59);
        complexGaussianDist(data, base * 9, 0.01, 0.61);
        complexGaussianDist(data, base * 10, 0.01, 0.52);
```

```
        complexGaussianDist(data, base * 11, 0.01, 0.49);
        complexGaussianDist(data, base * 12, 0.01, 0.51);
        complexGaussianDist(data, base * 13, 0.01, 0.48);
        complexGaussianDist(data, base * 14, 0.01, 0.51);
        complexGaussianDist(data, base * 15, 0.01, 0.46);
        complexGaussianDist(data, base * 16, 0.01, 0.35);
        complexGaussianDist(data, base * 17, 0.01, 0.20);
        complexGaussianDist(data, base * 18, 0.01, 0.10);
        complexGaussianDist(data, base * 19, 0.01, 0.5);
        complexGaussianDist(data, base * 20, 0.01, 0.1);


        SF2Sample sample = newSimpleFFTSample(sf2, "Oboe", data, base);
        SF2Layer layer = newLayer(sf2, "Oboe", sample);
        SF2Region region = layer.getRegions().get(0);
        region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
        region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -6000);
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, -1000);
        region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
        region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, -100);
        region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 9500);
        return layer;

    }

    public static SF2Layer new_bassoon(SF2Soundbank sf2) {
        int x = 8;
        int fftsize = 4096 * x;
        double[] data = new double[fftsize * 2];
        double base = x * 15;

        complexGaussianDist(data, base * 2, 100, 40);
        complexGaussianDist(data, base * 4, 100, 20);

        complexGaussianDist(data, base * 1, 0.01, 0.53);
        complexGaussianDist(data, base * 2, 0.01, 5);
        complexGaussianDist(data, base * 3, 0.01, 0.51);
        complexGaussianDist(data, base * 4, 0.01, 0.48);
        complexGaussianDist(data, base * 5, 0.01, 1.49);
        complexGaussianDist(data, base * 6, 0.01, 0.51);
        complexGaussianDist(data, base * 7, 0.01, 0.50);
        complexGaussianDist(data, base * 8, 0.01, 0.59);
        complexGaussianDist(data, base * 9, 0.01, 0.61);
        complexGaussianDist(data, base * 10, 0.01, 0.52);
        complexGaussianDist(data, base * 11, 0.01, 0.49);
        complexGaussianDist(data, base * 12, 0.01, 0.51);
        complexGaussianDist(data, base * 13, 0.01, 0.48);
        complexGaussianDist(data, base * 14, 0.01, 0.51);
        complexGaussianDist(data, base * 15, 0.01, 0.46);
        complexGaussianDist(data, base * 16, 0.01, 0.35);
        complexGaussianDist(data, base * 17, 0.01, 0.20);
        complexGaussianDist(data, base * 18, 0.01, 0.10);
        complexGaussianDist(data, base * 19, 0.01, 0.5);
        complexGaussianDist(data, base * 20, 0.01, 0.1);


        SF2Sample sample = newSimpleFFTSample(sf2, "Flute", data, base);
        SF2Layer layer = newLayer(sf2, "Flute", sample);
        SF2Region region = layer.getRegions().get(0);
        region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
        region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -6000);
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, -1000);
```

```
        region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
        region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, -100);
        region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 9500);
        return layer;

    }

    public static SF2Layer new_clarinet(SF2Soundbank sf2) {
        int x = 8;
        int fftsize = 4096 * x;
        double[] data = new double[fftsize * 2];
        double base = x * 15;

        complexGaussianDist(data, base * 1, 0.001, 0.5);
        complexGaussianDist(data, base * 2, 0.001, 0.02);
        complexGaussianDist(data, base * 3, 0.001, 0.2);
        complexGaussianDist(data, base * 4, 0.01, 0.1);

        complexGaussianDist(data, base * 4, 100, 60);
        complexGaussianDist(data, base * 6, 100, 20);
        complexGaussianDist(data, base * 8, 100, 20);

        complexGaussianDist(data, base * 5, 0.001, 0.1);
        complexGaussianDist(data, base * 6, 0.001, 0.09);
        complexGaussianDist(data, base * 7, 0.001, 0.02);
        complexGaussianDist(data, base * 8, 0.005, 0.16);
        complexGaussianDist(data, base * 9, 0.005, 0.96);
        complexGaussianDist(data, base * 10, 0.01, 0.9);
        complexGaussianDist(data, base * 11, 0.08, 1.2);
        complexGaussianDist(data, base * 12, 0.08, 1.8);
        complexGaussianDist(data, base * 13, 0.08, 1.6);
        complexGaussianDist(data, base * 14, 0.08, 1.2);
        complexGaussianDist(data, base * 15, 0.08, 0.9);
        complexGaussianDist(data, base * 16, 0.08, 0.5);
        complexGaussianDist(data, base * 17, 0.08, 0.2);


        complexGaussianDist(data, base * 1, 10, 8);
        complexGaussianDist(data, base * 2, 10, 8);
        complexGaussianDist(data, base * 3, 10, 8);
        complexGaussianDist(data, base * 4, 10, 8);
        complexGaussianDist(data, base * 5, 10, 8);
        complexGaussianDist(data, base * 6, 20, 9);
        complexGaussianDist(data, base * 7, 20, 9);
        complexGaussianDist(data, base * 8, 20, 9);
        complexGaussianDist(data, base * 9, 20, 8);
        complexGaussianDist(data, base * 10, 30, 8);
        complexGaussianDist(data, base * 11, 30, 9);
        complexGaussianDist(data, base * 12, 30, 9);
        complexGaussianDist(data, base * 13, 30, 8);
        complexGaussianDist(data, base * 14, 30, 8);
        complexGaussianDist(data, base * 15, 30, 7);
        complexGaussianDist(data, base * 16, 30, 7);
        complexGaussianDist(data, base * 17, 30, 6);

        SF2Sample sample = newSimpleFFTSample(sf2, "Clarinet", data, base);
        SF2Layer layer = newLayer(sf2, "Clarinet", sample);
        SF2Region region = layer.getRegions().get(0);
        region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
        region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -6000);
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, -1000);
        region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
```

```
        region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, -100);
        region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 9500);
        return layer;

    }

    public static SF2Layer new_timpani(SF2Soundbank sf2) {

        double datab[];
        double datah[];

        // Make Bass Part
        {
            int fftlen = 4096 * 8;
            double[] data = new double[2 * fftlen];
            double base = 48;
            complexGaussianDist(data, base * 2, 0.2, 1);
            complexGaussianDist(data, base * 3, 0.2, 0.7);
            complexGaussianDist(data, base * 5, 10, 1);
            complexGaussianDist(data, base * 6, 9, 1);
            complexGaussianDist(data, base * 8, 15, 1);
            complexGaussianDist(data, base * 9, 18, 0.8);
            complexGaussianDist(data, base * 11, 21, 0.5);
            complexGaussianDist(data, base * 13, 28, 0.3);
            complexGaussianDist(data, base * 14, 22, 0.1);
            randomPhase(data, new Random(3049912));
            ifft(data);
            normalize(data, 0.5);
            data = realPart(data);

            double d_len = data.length;
            for (int i = 0; i < data.length; i++) {
                double g = (1.0 - (i / d_len));
                data[i] *= g * g;
            }
            fadeUp(data, 40);
            datab = data;
        }

        // Make treble part
        {
            int fftlen = 4096 * 4;
            double[] data = new double[2 * fftlen];
            Random random = new Random(3049912);
            for (int i = 0; i < data.length; i += 2) {
                data[i] = (2.0 * (random.nextDouble() - 0.5)) * 0.1;
            }
            fft(data);
            // Remove all negative frequency
            for (int i = fftlen / 2; i < data.length; i++)
                data[i] = 0;
            for (int i = 1024 * 4; i < 2048 * 4; i++)
                data[i] = 1.0 - (i - 4096) / 4096.0;
            for (int i = 0; i < 300; i++) {
                double g = (1.0 - (i / 300.0));
                data[i] *= 1.0 + 20 * g * g;
            }
            for (int i = 0; i < 24; i++)
                data[i] = 0;
            randomPhase(data, new Random(3049912));
            ifft(data);
            normalize(data, 0.9);
```

```
            data = realPart(data);
            double gain = 1.0;
            for (int i = 0; i < data.length; i++) {
                data[i] *= gain;
                gain *= 0.9998;
            }
            datah = data;
        }

        for (int i = 0; i < datah.length; i++)
            datab[i] += datah[i] * 0.02;

        normalize(datab, 0.9);

        SF2Sample sample = newSimpleDrumSample(sf2, "Timpani", datab);

        SF2Layer layer = new SF2Layer(sf2);
        layer.setName("Timpani");

        SF2GlobalRegion global = new SF2GlobalRegion();
        layer.setGlobalZone(global);
        sf2.addResource(layer);

        SF2LayerRegion region = new SF2LayerRegion();
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 12000);
        region.putInteger(SF2Region.GENERATOR_INITIALATTENUATION, -100);
        region.setSample(sample);
        layer.getRegions().add(region);

        return layer;
    }

    public static SF2Layer new_melodic_toms(SF2Soundbank sf2) {

        double datab[];
        double datah[];

        // Make Bass Part
        {
            int fftlen = 4096 * 4;
            double[] data = new double[2 * fftlen];
            complexGaussianDist(data, 30, 0.5, 1);
            randomPhase(data, new Random(3049912));
            ifft(data);
            normalize(data, 0.8);
            data = realPart(data);

            double d_len = data.length;
            for (int i = 0; i < data.length; i++)
                data[i] *= (1.0 - (i / d_len));
            datab = data;
        }

        // Make treble part
        {
            int fftlen = 4096 * 4;
            double[] data = new double[2 * fftlen];
            Random random = new Random(3049912);
            for (int i = 0; i < data.length; i += 2)
                data[i] = (2.0 * (random.nextDouble() - 0.5)) * 0.1;
            fft(data);
            // Remove all negative frequency
```

```
            for (int i = fftlen / 2; i < data.length; i++)
                data[i] = 0;
            for (int i = 1024 * 4; i < 2048 * 4; i++)
                data[i] = 1.0 - (i - 4096) / 4096.0;
            for (int i = 0; i < 200; i++) {
                double g = (1.0 - (i / 200.0));
                data[i] *= 1.0 + 20 * g * g;
            }
            for (int i = 0; i < 30; i++)
                data[i] = 0;
            randomPhase(data, new Random(3049912));
            ifft(data);
            normalize(data, 0.9);
            data = realPart(data);
            double gain = 1.0;
            for (int i = 0; i < data.length; i++) {
                data[i] *= gain;
                gain *= 0.9996;
            }
            datah = data;
        }

        for (int i = 0; i < datah.length; i++)
            datab[i] += datah[i] * 0.5;
        for (int i = 0; i < 5; i++)
            datab[i] *= i / 5.0;

        normalize(datab, 0.99);

        SF2Sample sample = newSimpleDrumSample(sf2, "Melodic Toms", datab);
        sample.setOriginalPitch(63);

        SF2Layer layer = new SF2Layer(sf2);
        layer.setName("Melodic Toms");

        SF2GlobalRegion global = new SF2GlobalRegion();
        layer.setGlobalZone(global);
        sf2.addResource(layer);

        SF2LayerRegion region = new SF2LayerRegion();
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 12000);
        //region.putInteger(SF2Region.GENERATOR_SCALETUNING, 0);
        region.putInteger(SF2Region.GENERATOR_INITIALATTENUATION, -100);
        region.setSample(sample);
        layer.getRegions().add(region);

        return layer;
    }

    public static SF2Layer new_reverse_cymbal(SF2Soundbank sf2) {
        double datah[];
        {
            int fftlen = 4096 * 4;
            double[] data = new double[2 * fftlen];
            Random random = new Random(3049912);
            for (int i = 0; i < data.length; i += 2)
                data[i] = (2.0 * (random.nextDouble() - 0.5));
            for (int i = fftlen / 2; i < data.length; i++)
                data[i] = 0;
            for (int i = 0; i < 100; i++)
                data[i] = 0;

```

```
            for (int i = 0; i < 512 * 2; i++) {
                double gain = (i / (512.0 * 2.0));
                data[i] = 1 - gain;
            }
            datah = data;
        }

        SF2Sample sample = newSimpleFFTSample(sf2, "Reverse Cymbal",
                datah, 100, 20);

        SF2Layer layer = new SF2Layer(sf2);
        layer.setName("Reverse Cymbal");

        SF2GlobalRegion global = new SF2GlobalRegion();
        layer.setGlobalZone(global);
        sf2.addResource(layer);

        SF2LayerRegion region = new SF2LayerRegion();
        region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -200);
        region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, -12000);
        region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, -1000);
        region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
        region.setSample(sample);
        layer.getRegions().add(region);

        return layer;
    }

    public static SF2Layer new_snare_drum(SF2Soundbank sf2) {

        double datab[];
        double datah[];

        // Make Bass Part
        {
            int fftlen = 4096 * 4;
            double[] data = new double[2 * fftlen];
            complexGaussianDist(data, 24, 0.5, 1);
            randomPhase(data, new Random(3049912));
            ifft(data);
            normalize(data, 0.5);
            data = realPart(data);

            double d_len = data.length;
            for (int i = 0; i < data.length; i++)
                data[i] *= (1.0 - (i / d_len));
            datab = data;
        }

        // Make treble part
        {
            int fftlen = 4096 * 4;
            double[] data = new double[2 * fftlen];
            Random random = new Random(3049912);
            for (int i = 0; i < data.length; i += 2)
                data[i] = (2.0 * (random.nextDouble() - 0.5)) * 0.1;
            fft(data);
            // Remove all negative frequency
            for (int i = fftlen / 2; i < data.length; i++)
                data[i] = 0;
            for (int i = 1024 * 4; i < 2048 * 4; i++)
```

```
            data[i] = 1.0 - (i - 4096) / 4096.0;
        for (int i = 0; i < 300; i++) {
            double g = (1.0 - (i / 300.0));
            data[i] *= 1.0 + 20 * g * g;
        }
        for (int i = 0; i < 24; i++)
            data[i] = 0;
        randomPhase(data, new Random(3049912));
        ifft(data);
        normalize(data, 0.9);
        data = realPart(data);
        double gain = 1.0;
        for (int i = 0; i < data.length; i++) {
            data[i] *= gain;
            gain *= 0.9998;
        }
        datah = data;
    }

    for (int i = 0; i < datah.length; i++)
        datab[i] += datah[i];
    for (int i = 0; i < 5; i++)
        datab[i] *= i / 5.0;

    SF2Sample sample = newSimpleDrumSample(sf2, "Snare Drum", datab);

    SF2Layer layer = new SF2Layer(sf2);
    layer.setName("Snare Drum");

    SF2GlobalRegion global = new SF2GlobalRegion();
    layer.setGlobalZone(global);
    sf2.addResource(layer);

    SF2LayerRegion region = new SF2LayerRegion();
    region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 12000);
    region.putInteger(SF2Region.GENERATOR_SCALETUNING, 0);
    region.putInteger(SF2Region.GENERATOR_INITIALATTENUATION, -100);
    region.setSample(sample);
    layer.getRegions().add(region);

    return layer;
}

public static SF2Layer new_bass_drum(SF2Soundbank sf2) {

    double datab[];
    double datah[];

    // Make Bass Part
    {
        int fftlen = 4096 * 4;
        double[] data = new double[2 * fftlen];
        complexGaussianDist(data, 1.8 * 5 + 1, 2, 1);
        complexGaussianDist(data, 1.8 * 9 + 1, 2, 1);
        randomPhase(data, new Random(3049912));
        ifft(data);
        normalize(data, 0.9);
        data = realPart(data);
        double d_len = data.length;
        for (int i = 0; i < data.length; i++)
            data[i] *= (1.0 - (i / d_len));
        datab = data;
```

```
        }

        // Make treble part
        {
            int fftlen = 4096;
            double[] data = new double[2 * fftlen];
            Random random = new Random(3049912);
            for (int i = 0; i < data.length; i += 2)
                data[i] = (2.0 * (random.nextDouble() - 0.5)) * 0.1;
            fft(data);
            // Remove all negative frequency
            for (int i = fftlen / 2; i < data.length; i++)
                data[i] = 0;
            for (int i = 1024; i < 2048; i++)
                data[i] = 1.0 - (i - 1024) / 1024.0;
            for (int i = 0; i < 512; i++)
                data[i] = 10 * i / 512.0;
            for (int i = 0; i < 10; i++)
                data[i] = 0;
            randomPhase(data, new Random(3049912));
            ifft(data);
            normalize(data, 0.9);
            data = realPart(data);
            double gain = 1.0;
            for (int i = 0; i < data.length; i++) {
                data[i] *= gain;
                gain *= 0.999;
            }
            datah = data;
        }

        for (int i = 0; i < datah.length; i++)
            datab[i] += datah[i] * 0.5;
        for (int i = 0; i < 5; i++)
            datab[i] *= i / 5.0;

        SF2Sample sample = newSimpleDrumSample(sf2, "Bass␣Drum", datab);

        SF2Layer layer = new SF2Layer(sf2);
        layer.setName("Bass␣Drum");

        SF2GlobalRegion global = new SF2GlobalRegion();
        layer.setGlobalZone(global);
        sf2.addResource(layer);

        SF2LayerRegion region = new SF2LayerRegion();
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 12000);
        region.putInteger(SF2Region.GENERATOR_SCALETUNING, 0);
        region.putInteger(SF2Region.GENERATOR_INITIALATTENUATION, -100);
        region.setSample(sample);
        layer.getRegions().add(region);

        return layer;
    }

    public static SF2Layer new_tom(SF2Soundbank sf2) {

        double datab[];
        double datah[];

        // Make Bass Part
        {
```

```
            int fftlen = 4096 * 4;
            double[] data = new double[2 * fftlen];
            complexGaussianDist(data, 30, 0.5, 1);
            randomPhase(data, new Random(3049912));
            ifft(data);
            normalize(data, 0.8);
            data = realPart(data);

            double d_len = data.length;
            for (int i = 0; i < data.length; i++)
                data[i] *= (1.0 - (i / d_len));
            datab = data;
        }

        // Make treble part
        {
            int fftlen = 4096 * 4;
            double[] data = new double[2 * fftlen];
            Random random = new Random(3049912);
            for (int i = 0; i < data.length; i += 2)
                data[i] = (2.0 * (random.nextDouble() - 0.5)) * 0.1;
            fft(data);
            // Remove all negative frequency
            for (int i = fftlen / 2; i < data.length; i++)
                data[i] = 0;
            for (int i = 1024 * 4; i < 2048 * 4; i++)
                data[i] = 1.0 - (i - 4096) / 4096.0;
            for (int i = 0; i < 200; i++) {
                double g = (1.0 - (i / 200.0));
                data[i] *= 1.0 + 20 * g * g;
            }
            for (int i = 0; i < 30; i++)
                data[i] = 0;
            randomPhase(data, new Random(3049912));
            ifft(data);
            normalize(data, 0.9);
            data = realPart(data);
            double gain = 1.0;
            for (int i = 0; i < data.length; i++) {
                data[i] *= gain;
                gain *= 0.9996;
            }
            datah = data;
        }

        for (int i = 0; i < datah.length; i++)
            datab[i] += datah[i] * 0.5;
        for (int i = 0; i < 5; i++)
            datab[i] *= i / 5.0;

        normalize(datab, 0.99);

        SF2Sample sample = newSimpleDrumSample(sf2, "Tom", datab);
        sample.setOriginalPitch(50);

        SF2Layer layer = new SF2Layer(sf2);
        layer.setName("Tom");

        SF2GlobalRegion global = new SF2GlobalRegion();
        layer.setGlobalZone(global);
        sf2.addResource(layer);

```

```
        SF2LayerRegion region = new SF2LayerRegion();
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 12000);
        //region.putInteger(SF2Region.GENERATOR_SCALETUNING, 0);
        region.putInteger(SF2Region.GENERATOR_INITIALATTENUATION, -100);
        region.setSample(sample);
        layer.getRegions().add(region);

        return layer;
    }

    public static SF2Layer new_closed_hihat(SF2Soundbank sf2) {
        double datah[];

        // Make treble part
        {
            int fftlen = 4096 * 4;
            double[] data = new double[2 * fftlen];
            Random random = new Random(3049912);
            for (int i = 0; i < data.length; i += 2)
                data[i] = (2.0 * (random.nextDouble() - 0.5)) * 0.1;
            fft(data);
            // Remove all negative frequency
            for (int i = fftlen / 2; i < data.length; i++)
                data[i] = 0;
            for (int i = 1024 * 4; i < 2048 * 4; i++)
                data[i] = 1.0 - (i - 4096) / 4096.0;
            for (int i = 0; i < 2048; i++)
                data[i] = 0.2 + 0.8 * (i / 2048.0);
            randomPhase(data, new Random(3049912));
            ifft(data);
            normalize(data, 0.9);
            data = realPart(data);
            double gain = 1.0;
            for (int i = 0; i < data.length; i++) {
                data[i] *= gain;
                gain *= 0.9996;
            }
            datah = data;
        }

        for (int i = 0; i < 5; i++)
            datah[i] *= i / 5.0;
        SF2Sample sample = newSimpleDrumSample(sf2, "Closed Hi-Hat", datah);

        SF2Layer layer = new SF2Layer(sf2);
        layer.setName("Closed Hi-Hat");

        SF2GlobalRegion global = new SF2GlobalRegion();
        layer.setGlobalZone(global);
        sf2.addResource(layer);

        SF2LayerRegion region = new SF2LayerRegion();
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 12000);
        region.putInteger(SF2Region.GENERATOR_SCALETUNING, 0);
        region.putInteger(SF2Region.GENERATOR_EXCLUSIVECLASS, 1);
        region.setSample(sample);
        layer.getRegions().add(region);

        return layer;
    }

    public static SF2Layer new_open_hihat(SF2Soundbank sf2) {
```

```
        double datah[];
        {
            int fftlen = 4096 * 4;
            double[] data = new double[2 * fftlen];
            Random random = new Random(3049912);
            for (int i = 0; i < data.length; i += 2)
                data[i] = (2.0 * (random.nextDouble() - 0.5));
            for (int i = fftlen / 2; i < data.length; i++)
                data[i] = 0;
            for (int i = 0; i < 200; i++)
                data[i] = 0;
            for (int i = 0; i < 2048 * 4; i++) {
                double gain = (i / (2048.0 * 4.0));
                data[i] = gain;
            }
            datah = data;
        }

        SF2Sample sample = newSimpleFFTSample(sf2, "Open Hi-Hat", datah, 1000, 5);

        SF2Layer layer = new SF2Layer(sf2);
        layer.setName("Open Hi-Hat");

        SF2GlobalRegion global = new SF2GlobalRegion();
        layer.setGlobalZone(global);
        sf2.addResource(layer);

        SF2LayerRegion region = new SF2LayerRegion();
        region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 1500);
        region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 1500);
        region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
        region.putInteger(SF2Region.GENERATOR_SCALETUNING, 0);
        region.putInteger(SF2Region.GENERATOR_EXCLUSIVECLASS, 1);
        region.setSample(sample);
        layer.getRegions().add(region);

        return layer;
    }

    public static SF2Layer new_crash_cymbal(SF2Soundbank sf2) {
        double datah[];
        {
            int fftlen = 4096 * 4;
            double[] data = new double[2 * fftlen];
            Random random = new Random(3049912);
            for (int i = 0; i < data.length; i += 2)
                data[i] = (2.0 * (random.nextDouble() - 0.5));
            for (int i = fftlen / 2; i < data.length; i++)
                data[i] = 0;
            for (int i = 0; i < 100; i++)
                data[i] = 0;
            for (int i = 0; i < 512 * 2; i++) {
                double gain = (i / (512.0 * 2.0));
                data[i] = gain;
            }
            datah = data;
        }

        SF2Sample sample = newSimpleFFTSample(sf2, "Crash Cymbal", datah, 1000, 5);

        SF2Layer layer = new SF2Layer(sf2);
```

```
2355        layer.setName("Crash␣Cymbal");
2356
2357        SF2GlobalRegion global = new SF2GlobalRegion();
2358        layer.setGlobalZone(global);
2359        sf2.addResource(layer);
2360
2361        SF2LayerRegion region = new SF2LayerRegion();
2362        region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 1800);
2363        region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
2364        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 1800);
2365        region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
2366        region.putInteger(SF2Region.GENERATOR_SCALETUNING, 0);
2367        region.setSample(sample);
2368        layer.getRegions().add(region);
2369
2370        return layer;
2371    }
2372
2373    public static SF2Layer new_side_stick(SF2Soundbank sf2) {
2374        double datab[];
2375
2376        // Make treble part
2377        {
2378            int fftlen = 4096 * 4;
2379            double[] data = new double[2 * fftlen];
2380            Random random = new Random(3049912);
2381            for (int i = 0; i < data.length; i += 2)
2382                data[i] = (2.0 * (random.nextDouble() - 0.5)) * 0.1;
2383            fft(data);
2384            // Remove all negative frequency
2385            for (int i = fftlen / 2; i < data.length; i++)
2386                data[i] = 0;
2387            for (int i = 1024 * 4; i < 2048 * 4; i++)
2388                data[i] = 1.0 - (i - 4096) / 4096.0;
2389            for (int i = 0; i < 200; i++) {
2390                double g = (1.0 - (i / 200.0));
2391                data[i] *= 1.0 + 20 * g * g;
2392            }
2393            for (int i = 0; i < 30; i++)
2394                data[i] = 0;
2395            randomPhase(data, new Random(3049912));
2396            ifft(data);
2397            normalize(data, 0.9);
2398            data = realPart(data);
2399            double gain = 1.0;
2400            for (int i = 0; i < data.length; i++) {
2401                data[i] *= gain;
2402                gain *= 0.9996;
2403            }
2404            datab = data;
2405        }
2406
2407        for (int i = 0; i < 10; i++)
2408            datab[i] *= i / 10.0;
2409
2410        SF2Sample sample = newSimpleDrumSample(sf2, "Side␣Stick", datab);
2411
2412        SF2Layer layer = new SF2Layer(sf2);
2413        layer.setName("Side␣Stick");
2414
2415        SF2GlobalRegion global = new SF2GlobalRegion();
2416        layer.setGlobalZone(global);
```

```
        sf2.addResource(layer);

        SF2LayerRegion region = new SF2LayerRegion();
        region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 12000);
        region.putInteger(SF2Region.GENERATOR_SCALETUNING, 0);
        region.putInteger(SF2Region.GENERATOR_INITIALATTENUATION, -50);
        region.setSample(sample);
        layer.getRegions().add(region);

        return layer;

    }

    public static SF2Sample newSimpleFFTSample(SF2Soundbank sf2, String name,
            double[] data, double base) {
        return newSimpleFFTSample(sf2, name, data, base, 10);
    }

    public static SF2Sample newSimpleFFTSample(SF2Soundbank sf2, String name,
            double[] data, double base, int fadeuptime) {

        int fftsize = data.length / 2;
        AudioFormat format = new AudioFormat(44100, 16, 1, true, false);
        double basefreq = (base / fftsize) * format.getSampleRate() * 0.5;

        randomPhase(data);
        ifft(data);
        data = realPart(data);
        normalize(data, 0.9);
        float[] fdata = toFloat(data);
        fdata = loopExtend(fdata, fdata.length + 512);
        fadeUp(fdata, fadeuptime);
        byte[] bdata = toBytes(fdata, format);

        /*
         * Create SoundFont2 sample.
         */
        SF2Sample sample = new SF2Sample(sf2);
        sample.setName(name);
        sample.setData(bdata);
        sample.setStartLoop(256);
        sample.setEndLoop(fftsize + 256);
        sample.setSampleRate((long) format.getSampleRate());
        double orgnote = (69 + 12)
                + (12 * Math.log(basefreq / 440.0) / Math.log(2));
        sample.setOriginalPitch((int) orgnote);
        sample.setPitchCorrection((byte) (-(orgnote - (int) orgnote) * 100.0));
        sf2.addResource(sample);

        return sample;
    }

    public static SF2Sample newSimpleFFTSample_dist(SF2Soundbank sf2,
            String name, double[] data, double base, double preamp) {

        int fftsize = data.length / 2;
        AudioFormat format = new AudioFormat(44100, 16, 1, true, false);
        double basefreq = (base / fftsize) * format.getSampleRate() * 0.5;

        randomPhase(data);
        ifft(data);
        data = realPart(data);
```

```
2479
2480        for (int i = 0; i < data.length; i++) {
2481            data[i] = (1 - Math.exp(-Math.abs(data[i] * preamp)))
2482                    * Math.signum(data[i]);
2483        }
2484
2485        normalize(data, 0.9);
2486        float[] fdata = toFloat(data);
2487        fdata = loopExtend(fdata, fdata.length + 512);
2488        fadeUp(fdata, 80);
2489        byte[] bdata = toBytes(fdata, format);
2490
2491        /*
2492         * Create SoundFont2 sample.
2493         */
2494        SF2Sample sample = new SF2Sample(sf2);
2495        sample.setName(name);
2496        sample.setData(bdata);
2497        sample.setStartLoop(256);
2498        sample.setEndLoop(fftsize + 256);
2499        sample.setSampleRate((long) format.getSampleRate());
2500        double orgnote = (69 + 12)
2501                + (12 * Math.log(basefreq / 440.0) / Math.log(2));
2502        sample.setOriginalPitch((int) orgnote);
2503        sample.setPitchCorrection((byte) (-(orgnote - (int) orgnote) * 100.0));
2504        sf2.addResource(sample);
2505
2506        return sample;
2507    }
2508
2509    public static SF2Sample newSimpleDrumSample(SF2Soundbank sf2, String name,
2510            double[] data) {
2511
2512        int fftsize = data.length;
2513        AudioFormat format = new AudioFormat(44100, 16, 1, true, false);
2514
2515        byte[] bdata = toBytes(toFloat(realPart(data)), format);
2516
2517        /*
2518         * Create SoundFont2 sample.
2519         */
2520        SF2Sample sample = new SF2Sample(sf2);
2521        sample.setName(name);
2522        sample.setData(bdata);
2523        sample.setStartLoop(256);
2524        sample.setEndLoop(fftsize + 256);
2525        sample.setSampleRate((long) format.getSampleRate());
2526        sample.setOriginalPitch(60);
2527        sf2.addResource(sample);
2528
2529        return sample;
2530    }
2531
2532    public static SF2Layer newLayer(SF2Soundbank sf2, String name, SF2Sample sample) {
2533        SF2LayerRegion region = new SF2LayerRegion();
2534        region.setSample(sample);
2535
2536        SF2Layer layer = new SF2Layer(sf2);
2537        layer.setName(name);
2538        layer.getRegions().add(region);
2539        sf2.addResource(layer);
2540
```

```
        return layer;
    }

    public static SF2Instrument newInstrument(SF2Soundbank sf2, String name,
            Patch patch, SF2Layer... layers) {

        /*
         * Create SoundFont2 instrument.
         */
        SF2Instrument ins = new SF2Instrument(sf2);
        ins.setPatch(patch);
        ins.setName(name);
        sf2.addInstrument(ins);

        /*
         * Create region for instrument.
         */
        for (int i = 0; i < layers.length; i++) {
            SF2InstrumentRegion insregion = new SF2InstrumentRegion();
            insregion.setLayer(layers[i]);
            ins.getRegions().add(insregion);
        }

        return ins;
    }

    static public void ifft(double[] data) {
        new FFT(data.length / 2, 1).transform(data);
    }

    static public void fft(double[] data) {
        new FFT(data.length / 2, -1).transform(data);
    }

    public static void complexGaussianDist(double[] cdata, double m,
            double s, double v) {
        for (int x = 0; x < cdata.length / 4; x++) {
            cdata[x * 2] += v * (1.0 / (s * Math.sqrt(2 * Math.PI))
                    * Math.exp((-1.0 / 2.0) * Math.pow((x - m) / s, 2.0)));
        }
    }

    static public void randomPhase(double[] data) {
        for (int i = 0; i < data.length; i += 2) {
            double phase = Math.random() * 2 * Math.PI;
            double d = data[i];
            data[i] = Math.sin(phase) * d;
            data[i + 1] = Math.cos(phase) * d;
        }
    }

    static public void randomPhase(double[] data, Random random) {
        for (int i = 0; i < data.length; i += 2) {
            double phase = random.nextDouble() * 2 * Math.PI;
            double d = data[i];
            data[i] = Math.sin(phase) * d;
            data[i + 1] = Math.cos(phase) * d;
        }
    }

    static public void normalize(double[] data, double target) {
        double maxvalue = 0;
```

```
        for (int i = 0; i < data.length; i++) {
            if (data[i] > maxvalue)
                maxvalue = data[i];
            if (-data[i] > maxvalue)
                maxvalue = -data[i];
        }
        if (maxvalue == 0)
            return;
        double gain = target / maxvalue;
        for (int i = 0; i < data.length; i++)
            data[i] *= gain;
    }

    static public void normalize(float[] data, double target) {
        double maxvalue = 0.5;
        for (int i = 0; i < data.length; i++) {
            if (data[i * 2] > maxvalue)
                maxvalue = data[i * 2];
            if (-data[i * 2] > maxvalue)
                maxvalue = -data[i * 2];
        }
        double gain = target / maxvalue;
        for (int i = 0; i < data.length; i++)
            data[i * 2] *= gain;
    }

    static public double[] realPart(double[] in) {
        double[] out = new double[in.length / 2];
        for (int i = 0; i < out.length; i++) {
            out[i] = in[i * 2];
        }
        return out;
    }

    static public double[] imgPart(double[] in) {
        double[] out = new double[in.length / 2];
        for (int i = 0; i < out.length; i++) {
            out[i] = in[i * 2];
        }
        return out;
    }

    static public float[] toFloat(double[] in) {
        float[] out = new float[in.length];
        for (int i = 0; i < out.length; i++) {
            out[i] = (float) in[i];
        }
        return out;
    }

    static public byte[] toBytes(float[] in, AudioFormat format) {
        byte[] out = new byte[in.length * format.getFrameSize()];
        return AudioFloatConverter.getConverter(format).toByteArray(in, out);
    }

    static public void fadeUp(double[] data, int samples) {
        double dsamples = samples;
        for (int i = 0; i < samples; i++)
            data[i] *= i / dsamples;
    }

    static public void fadeUp(float[] data, int samples) {
```

```
        double dsamples = samples;
        for (int i = 0; i < samples; i++)
            data[i] *= i / dsamples;
    }

    static public double[] loopExtend(double[] data, int newsize) {
        double[] outdata = new double[newsize];
        int p_len = data.length;
        int p_ps = 0;
        for (int i = 0; i < outdata.length; i++) {
            outdata[i] = data[p_ps];
            p_ps++;
            if (p_ps == p_len)
                p_ps = 0;
        }
        return outdata;
    }

    static public float[] loopExtend(float[] data, int newsize) {
        float[] outdata = new float[newsize];
        int p_len = data.length;
        int p_ps = 0;
        for (int i = 0; i < outdata.length; i++) {
            outdata[i] = data[p_ps];
            p_ps++;
            if (p_ps == p_len)
                p_ps = 0;
        }
        return outdata;
    }
}
```

32 com/sun/media/sound/FFT.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * Fast Fourier Transformer.
 *
 * @author Karl Helgason
 */
public final class FFT {

    private double[] w;
    private int fftFrameSize;
    private int sign;
    private int[] bitm_array;
    private int fftFrameSize2;

    // Sign = -1 is FFT, 1 is IFFT (inverse FFT)
    // Data = Interlaced double array to be transformed.
    // The order is: real (sin), complex (cos)
    // Framesize must be power of 2
    public FFT(int fftFrameSize, int sign) {
        w = computeTwiddleFactors(fftFrameSize, sign);

        this.fftFrameSize = fftFrameSize;
        this.sign = sign;
        fftFrameSize2 = fftFrameSize << 1;

        // Pre-process Bit-Reversal
        bitm_array = new int[fftFrameSize2];
        for (int i = 2; i < fftFrameSize2; i += 2) {
            int j;
            int bitm;
            for (bitm = 2, j = 0; bitm < fftFrameSize2; bitm <<= 1) {
                if ((i & bitm) != 0)
                    j++;
                j <<= 1;
            }
```

```
            bitm_array[i] = j;
        }

    }

    public void transform(double[] data) {
        bitreversal(data);
        calc(fftFrameSize, data, sign, w);
    }

    private final static double[] computeTwiddleFactors(int fftFrameSize,
            int sign) {

        int imax = (int) (Math.log(fftFrameSize) / Math.log(2.));

        double[] warray = new double[(fftFrameSize - 1) * 4];
        int w_index = 0;

        for (int i = 0,  nstep = 2; i < imax; i++) {
            int jmax = nstep;
            nstep <<= 1;

            double wr = 1.0;
            double wi = 0.0;

            double arg = Math.PI / (jmax >> 1);
            double wfr = Math.cos(arg);
            double wfi = sign * Math.sin(arg);

            for (int j = 0; j < jmax; j += 2) {
                warray[w_index++] = wr;
                warray[w_index++] = wi;

                double tempr = wr;
                wr = tempr * wfr - wi * wfi;
                wi = tempr * wfi + wi * wfr;
            }
        }

        // PRECOMPUTATION of wwr1, wwi1 for factor 4 Decomposition (3 * complex
        // operators and 8 +/- complex operators)
        {
            w_index = 0;
            int w_index2 = warray.length >> 1;
            for (int i = 0,  nstep = 2; i < (imax - 1); i++) {
                int jmax = nstep;
                nstep *= 2;

                int ii = w_index + jmax;
                for (int j = 0; j < jmax; j += 2) {
                    double wr = warray[w_index++];
                    double wi = warray[w_index++];
                    double wr1 = warray[ii++];
                    double wi1 = warray[ii++];
                    warray[w_index2++] = wr * wr1 - wi * wi1;
                    warray[w_index2++] = wr * wi1 + wi * wr1;
                }
            }

        }

        return warray;
```

```
123     }
124
125     private final static void calc(int fftFrameSize, double[] data, int sign,
126             double[] w) {
127
128         final int fftFrameSize2 = fftFrameSize << 1;
129
130         int nstep = 2;
131
132         if (nstep >= fftFrameSize2)
133             return;
134         int i = nstep - 2;
135         if (sign == -1)
136             calcF4F(fftFrameSize, data, i, nstep, w);
137         else
138             calcF4I(fftFrameSize, data, i, nstep, w);
139
140     }
141
142     private final static void calcF2E(int fftFrameSize, double[] data, int i,
143             int nstep, double[] w) {
144         int jmax = nstep;
145         for (int n = 0; n < jmax; n += 2) {
146             double wr = w[i++];
147             double wi = w[i++];
148             int m = n + jmax;
149             double datam_r = data[m];
150             double datam_i = data[m + 1];
151             double datan_r = data[n];
152             double datan_i = data[n + 1];
153             double tempr = datam_r * wr - datam_i * wi;
154             double tempi = datam_r * wi + datam_i * wr;
155             data[m] = datan_r - tempr;
156             data[m + 1] = datan_i - tempi;
157             data[n] = datan_r + tempr;
158             data[n + 1] = datan_i + tempi;
159         }
160         return;
161
162     }
163
164     // Perform Factor-4 Decomposition with 3 * complex operators and 8 +/-
165     // complex operators
166     private final static void calcF4F(int fftFrameSize, double[] data, int i,
167             int nstep, double[] w) {
168         final int fftFrameSize2 = fftFrameSize << 1; // 2*fftFrameSize;
169         // Factor-4 Decomposition
170
171         int w_len = w.length >> 1;
172         while (nstep < fftFrameSize2) {
173
174             if (nstep << 2 == fftFrameSize2) {
175                 // Goto Factor-4 Final Decomposition
176                 // calcF4E(data, i, nstep, -1, w);
177                 calcF4FE(fftFrameSize, data, i, nstep, w);
178                 return;
179             }
180             int jmax = nstep;
181             int nnstep = nstep << 1;
182             if (nnstep == fftFrameSize2) {
183                 // Factor-4 Decomposition not possible
184                 calcF2E(fftFrameSize, data, i, nstep, w);
```

```
            return;
        }
        nstep <<= 2;
        int ii = i + jmax;
        int iii = i + w_len;

        {
            i += 2;
            ii += 2;
            iii += 2;

            for (int n = 0; n < fftFrameSize2; n += nstep) {
                int m = n + jmax;

                double datam1_r = data[m];
                double datam1_i = data[m + 1];
                double datan1_r = data[n];
                double datan1_i = data[n + 1];

                n += nnstep;
                m += nnstep;
                double datam2_r = data[m];
                double datam2_i = data[m + 1];
                double datan2_r = data[n];
                double datan2_i = data[n + 1];

                double tempr = datam1_r;
                double tempi = datam1_i;

                datam1_r = datan1_r - tempr;
                datam1_i = datan1_i - tempi;
                datan1_r = datan1_r + tempr;
                datan1_i = datan1_i + tempi;

                double n2w1r = datan2_r;
                double n2w1i = datan2_i;
                double m2ww1r = datam2_r;
                double m2ww1i = datam2_i;

                tempr = m2ww1r - n2w1r;
                tempi = m2ww1i - n2w1i;

                datam2_r = datam1_r + tempi;
                datam2_i = datam1_i - tempr;
                datam1_r = datam1_r - tempi;
                datam1_i = datam1_i + tempr;

                tempr = n2w1r + m2ww1r;
                tempi = n2w1i + m2ww1i;

                datan2_r = datan1_r - tempr;
                datan2_i = datan1_i - tempi;
                datan1_r = datan1_r + tempr;
                datan1_i = datan1_i + tempi;

                data[m] = datam2_r;
                data[m + 1] = datam2_i;
                data[n] = datan2_r;
                data[n + 1] = datan2_i;

                n -= nnstep;
                m -= nnstep;
```

```
247                     data[m] = datam1_r;
248                     data[m + 1] = datam1_i;
249                     data[n] = datan1_r;
250                     data[n + 1] = datan1_i;
251 
252                 }
253             }
254 
255             for (int j = 2; j < jmax; j += 2) {
256                 double wr = w[i++];
257                 double wi = w[i++];
258                 double wr1 = w[ii++];
259                 double wi1 = w[ii++];
260                 double wwr1 = w[iii++];
261                 double wwi1 = w[iii++];
262                 // double wwr1 = wr * wr1 - wi * wi1; // these numbers can be
263                 // precomputed!!!
264                 // double wwi1 = wr * wi1 + wi * wr1;
265 
266                 for (int n = j; n < fftFrameSize2; n += nstep) {
267                     int m = n + jmax;
268 
269                     double datam1_r = data[m];
270                     double datam1_i = data[m + 1];
271                     double datan1_r = data[n];
272                     double datan1_i = data[n + 1];
273 
274                     n += nnstep;
275                     m += nnstep;
276                     double datam2_r = data[m];
277                     double datam2_i = data[m + 1];
278                     double datan2_r = data[n];
279                     double datan2_i = data[n + 1];
280 
281                     double tempr = datam1_r * wr - datam1_i * wi;
282                     double tempi = datam1_r * wi + datam1_i * wr;
283 
284                     datam1_r = datan1_r - tempr;
285                     datam1_i = datan1_i - tempi;
286                     datan1_r = datan1_r + tempr;
287                     datan1_i = datan1_i + tempi;
288 
289                     double n2w1r = datan2_r * wr1 - datan2_i * wi1;
290                     double n2w1i = datan2_r * wi1 + datan2_i * wr1;
291                     double m2ww1r = datam2_r * wwr1 - datam2_i * wwi1;
292                     double m2ww1i = datam2_r * wwi1 + datam2_i * wwr1;
293 
294                     tempr = m2ww1r - n2w1r;
295                     tempi = m2ww1i - n2w1i;
296 
297                     datam2_r = datam1_r + tempi;
298                     datam2_i = datam1_i - tempr;
299                     datam1_r = datam1_r - tempi;
300                     datam1_i = datam1_i + tempr;
301 
302                     tempr = n2w1r + m2ww1r;
303                     tempi = n2w1i + m2ww1i;
304 
305                     datan2_r = datan1_r - tempr;
306                     datan2_i = datan1_i - tempi;
307                     datan1_r = datan1_r + tempr;
308                     datan1_i = datan1_i + tempi;
```

```
309
310                     data[m] = datam2_r;
311                     data[m + 1] = datam2_i;
312                     data[n] = datan2_r;
313                     data[n + 1] = datan2_i;
314
315                     n -= nnstep;
316                     m -= nnstep;
317                     data[m] = datam1_r;
318                     data[m + 1] = datam1_i;
319                     data[n] = datan1_r;
320                     data[n + 1] = datan1_i;
321                 }
322             }
323
324             i += jmax << 1;
325
326         }
327
328         calcF2E(fftFrameSize, data, i, nstep, w);
329
330     }
331
332     // Perform Factor-4 Decomposition with 3 * complex operators and 8 +/-
333     // complex operators
334     private final static void calcF4I(int fftFrameSize, double[] data, int i,
335             int nstep, double[] w) {
336         final int fftFrameSize2 = fftFrameSize << 1; // 2*fftFrameSize;
337         // Factor-4 Decomposition
338
339         int w_len = w.length >> 1;
340         while (nstep < fftFrameSize2) {
341
342             if (nstep << 2 == fftFrameSize2) {
343                 // Goto Factor-4 Final Decomposition
344                 // calcF4E(data, i, nstep, 1, w);
345                 calcF4IE(fftFrameSize, data, i, nstep, w);
346                 return;
347             }
348             int jmax = nstep;
349             int nnstep = nstep << 1;
350             if (nnstep == fftFrameSize2) {
351                 // Factor-4 Decomposition not possible
352                 calcF2E(fftFrameSize, data, i, nstep, w);
353                 return;
354             }
355             nstep <<= 2;
356             int ii = i + jmax;
357             int iii = i + w_len;
358             {
359                 i += 2;
360                 ii += 2;
361                 iii += 2;
362
363                 for (int n = 0; n < fftFrameSize2; n += nstep) {
364                     int m = n + jmax;
365
366                     double datam1_r = data[m];
367                     double datam1_i = data[m + 1];
368                     double datan1_r = data[n];
369                     double datan1_i = data[n + 1];
370
```

```
                n += nnstep;
                m += nnstep;
                double datam2_r = data[m];
                double datam2_i = data[m + 1];
                double datan2_r = data[n];
                double datan2_i = data[n + 1];

                double tempr = datam1_r;
                double tempi = datam1_i;

                datam1_r = datan1_r - tempr;
                datam1_i = datan1_i - tempi;
                datan1_r = datan1_r + tempr;
                datan1_i = datan1_i + tempi;

                double n2w1r = datan2_r;
                double n2w1i = datan2_i;
                double m2ww1r = datam2_r;
                double m2ww1i = datam2_i;

                tempr = n2w1r - m2ww1r;
                tempi = n2w1i - m2ww1i;

                datam2_r = datam1_r + tempi;
                datam2_i = datam1_i - tempr;
                datam1_r = datam1_r - tempi;
                datam1_i = datam1_i + tempr;

                tempr = n2w1r + m2ww1r;
                tempi = n2w1i + m2ww1i;

                datan2_r = datan1_r - tempr;
                datan2_i = datan1_i - tempi;
                datan1_r = datan1_r + tempr;
                datan1_i = datan1_i + tempi;

                data[m] = datam2_r;
                data[m + 1] = datam2_i;
                data[n] = datan2_r;
                data[n + 1] = datan2_i;

                n -= nnstep;
                m -= nnstep;
                data[m] = datam1_r;
                data[m + 1] = datam1_i;
                data[n] = datan1_r;
                data[n + 1] = datan1_i;

            }

        }
        for (int j = 2; j < jmax; j += 2) {
            double wr = w[i++];
            double wi = w[i++];
            double wr1 = w[ii++];
            double wi1 = w[ii++];
            double wwr1 = w[iii++];
            double wwi1 = w[iii++];
            // double wwr1 = wr * wr1 - wi * wi1; // these numbers can be
            // precomputed!!!
            // double wwi1 = wr * wi1 + wi * wr1;

```

```
433                 for (int n = j; n < fftFrameSize2; n += nstep) {
434                     int m = n + jmax;
435 
436                     double datam1_r = data[m];
437                     double datam1_i = data[m + 1];
438                     double datan1_r = data[n];
439                     double datan1_i = data[n + 1];
440 
441                     n += nnstep;
442                     m += nnstep;
443                     double datam2_r = data[m];
444                     double datam2_i = data[m + 1];
445                     double datan2_r = data[n];
446                     double datan2_i = data[n + 1];
447 
448                     double tempr = datam1_r * wr - datam1_i * wi;
449                     double tempi = datam1_r * wi + datam1_i * wr;
450 
451                     datam1_r = datan1_r - tempr;
452                     datam1_i = datan1_i - tempi;
453                     datan1_r = datan1_r + tempr;
454                     datan1_i = datan1_i + tempi;
455 
456                     double n2w1r = datan2_r * wr1 - datan2_i * wi1;
457                     double n2w1i = datan2_r * wi1 + datan2_i * wr1;
458                     double m2ww1r = datam2_r * wwr1 - datam2_i * wwi1;
459                     double m2ww1i = datam2_r * wwi1 + datam2_i * wwr1;
460 
461                     tempr = n2w1r - m2ww1r;
462                     tempi = n2w1i - m2ww1i;
463 
464                     datam2_r = datam1_r + tempi;
465                     datam2_i = datam1_i - tempr;
466                     datam1_r = datam1_r - tempi;
467                     datam1_i = datam1_i + tempr;
468 
469                     tempr = n2w1r + m2ww1r;
470                     tempi = n2w1i + m2ww1i;
471 
472                     datan2_r = datan1_r - tempr;
473                     datan2_i = datan1_i - tempi;
474                     datan1_r = datan1_r + tempr;
475                     datan1_i = datan1_i + tempi;
476 
477                     data[m] = datam2_r;
478                     data[m + 1] = datam2_i;
479                     data[n] = datan2_r;
480                     data[n + 1] = datan2_i;
481 
482                     n -= nnstep;
483                     m -= nnstep;
484                     data[m] = datam1_r;
485                     data[m + 1] = datam1_i;
486                     data[n] = datan1_r;
487                     data[n + 1] = datan1_i;
488 
489                 }
490             }
491 
492             i += jmax << 1;
493 
494         }
```

```
495
496         calcF2E(fftFrameSize, data, i, nstep, w);
497
498     }
499
500     // Perform Factor-4 Decomposition with 3 * complex operators and 8 +/-
501     // complex operators
502     private final static void calcF4FE(int fftFrameSize, double[] data, int i,
503             int nstep, double[] w) {
504         final int fftFrameSize2 = fftFrameSize << 1; // 2*fftFrameSize;
505         // Factor-4 Decomposition
506
507         int w_len = w.length >> 1;
508         while (nstep < fftFrameSize2) {
509
510             int jmax = nstep;
511             int nnstep = nstep << 1;
512             if (nnstep == fftFrameSize2) {
513                 // Factor-4 Decomposition not possible
514                 calcF2E(fftFrameSize, data, i, nstep, w);
515                 return;
516             }
517             nstep <<= 2;
518             int ii = i + jmax;
519             int iii = i + w_len;
520             for (int n = 0; n < jmax; n += 2) {
521                 double wr = w[i++];
522                 double wi = w[i++];
523                 double wr1 = w[ii++];
524                 double wi1 = w[ii++];
525                 double wwr1 = w[iii++];
526                 double wwi1 = w[iii++];
527                 // double wwr1 = wr * wr1 - wi * wi1; // these numbers can be
528                 // precomputed!!!
529                 // double wwi1 = wr * wi1 + wi * wr1;
530
531                 int m = n + jmax;
532
533                 double datam1_r = data[m];
534                 double datam1_i = data[m + 1];
535                 double datan1_r = data[n];
536                 double datan1_i = data[n + 1];
537
538                 n += nnstep;
539                 m += nnstep;
540                 double datam2_r = data[m];
541                 double datam2_i = data[m + 1];
542                 double datan2_r = data[n];
543                 double datan2_i = data[n + 1];
544
545                 double tempr = datam1_r * wr - datam1_i * wi;
546                 double tempi = datam1_r * wi + datam1_i * wr;
547
548                 datam1_r = datan1_r - tempr;
549                 datam1_i = datan1_i - tempi;
550                 datan1_r = datan1_r + tempr;
551                 datan1_i = datan1_i + tempi;
552
553                 double n2w1r = datan2_r * wr1 - datan2_i * wi1;
554                 double n2w1i = datan2_r * wi1 + datan2_i * wr1;
555                 double m2ww1r = datam2_r * wwr1 - datam2_i * wwi1;
556                 double m2ww1i = datam2_r * wwi1 + datam2_i * wwr1;
```

```
                tempr = m2ww1r - n2w1r;
                tempi = m2ww1i - n2w1i;

                datam2_r = datam1_r + tempi;
                datam2_i = datam1_i - tempr;
                datam1_r = datam1_r - tempi;
                datam1_i = datam1_i + tempr;

                tempr = n2w1r + m2ww1r;
                tempi = n2w1i + m2ww1i;

                datan2_r = datan1_r - tempr;
                datan2_i = datan1_i - tempi;
                datan1_r = datan1_r + tempr;
                datan1_i = datan1_i + tempi;

                data[m] = datam2_r;
                data[m + 1] = datam2_i;
                data[n] = datan2_r;
                data[n + 1] = datan2_i;

                n -= nnstep;
                m -= nnstep;
                data[m] = datam1_r;
                data[m + 1] = datam1_i;
                data[n] = datan1_r;
                data[n + 1] = datan1_i;

            }

            i += jmax << 1;

        }

    }

    // Perform Factor-4 Decomposition with 3 * complex operators and 8 +/-
    // complex operators
    private final static void calcF4IE(int fftFrameSize, double[] data, int i,
            int nstep, double[] w) {
        final int fftFrameSize2 = fftFrameSize << 1; // 2*fftFrameSize;
        // Factor-4 Decomposition

        int w_len = w.length >> 1;
        while (nstep < fftFrameSize2) {

            int jmax = nstep;
            int nnstep = nstep << 1;
            if (nnstep == fftFrameSize2) {
                // Factor-4 Decomposition not possible
                calcF2E(fftFrameSize, data, i, nstep, w);
                return;
            }
            nstep <<= 2;
            int ii = i + jmax;
            int iii = i + w_len;
            for (int n = 0; n < jmax; n += 2) {
                double wr = w[i++];
                double wi = w[i++];
                double wr1 = w[ii++];
                double wi1 = w[ii++];
```

```
            double wwr1 = w[iii++];
            double wwi1 = w[iii++];
            // double wwr1 = wr * wr1 - wi * wi1; // these numbers can be
            // precomputed!!!
            // double wwi1 = wr * wi1 + wi * wr1;

            int m = n + jmax;

            double datam1_r = data[m];
            double datam1_i = data[m + 1];
            double datan1_r = data[n];
            double datan1_i = data[n + 1];

            n += nnstep;
            m += nnstep;
            double datam2_r = data[m];
            double datam2_i = data[m + 1];
            double datan2_r = data[n];
            double datan2_i = data[n + 1];

            double tempr = datam1_r * wr - datam1_i * wi;
            double tempi = datam1_r * wi + datam1_i * wr;

            datam1_r = datan1_r - tempr;
            datam1_i = datan1_i - tempi;
            datan1_r = datan1_r + tempr;
            datan1_i = datan1_i + tempi;

            double n2w1r = datan2_r * wr1 - datan2_i * wi1;
            double n2w1i = datan2_r * wi1 + datan2_i * wr1;
            double m2ww1r = datam2_r * wwr1 - datam2_i * wwi1;
            double m2ww1i = datam2_r * wwi1 + datam2_i * wwr1;

            tempr = n2w1r - m2ww1r;
            tempi = n2w1i - m2ww1i;

            datam2_r = datam1_r + tempi;
            datam2_i = datam1_i - tempr;
            datam1_r = datam1_r - tempi;
            datam1_i = datam1_i + tempr;

            tempr = n2w1r + m2ww1r;
            tempi = n2w1i + m2ww1i;

            datan2_r = datan1_r - tempr;
            datan2_i = datan1_i - tempi;
            datan1_r = datan1_r + tempr;
            datan1_i = datan1_i + tempi;

            data[m] = datam2_r;
            data[m + 1] = datam2_i;
            data[n] = datan2_r;
            data[n + 1] = datan2_i;

            n -= nnstep;
            m -= nnstep;
            data[m] = datam1_r;
            data[m + 1] = datam1_i;
            data[n] = datan1_r;
            data[n + 1] = datan1_i;

        }
```

```
681
682             i += jmax << 1;
683
684         }
685
686     }
687
688     private final void bitreversal(double[] data) {
689         if (fftFrameSize < 4)
690             return;
691
692         int inverse = fftFrameSize2 - 2;
693         for (int i = 0; i < fftFrameSize; i += 4) {
694             int j = bitm_array[i];
695
696             // Performing Bit-Reversal, even v.s. even, O(2N)
697             if (i < j) {
698
699                 int n = i;
700                 int m = j;
701
702                 // COMPLEX: SWAP(data[n], data[m])
703                 // Real Part
704                 double tempr = data[n];
705                 data[n] = data[m];
706                 data[m] = tempr;
707                 // Imagery Part
708                 n++;
709                 m++;
710                 double tempi = data[n];
711                 data[n] = data[m];
712                 data[m] = tempi;
713
714                 n = inverse - i;
715                 m = inverse - j;
716
717                 // COMPLEX: SWAP(data[n], data[m])
718                 // Real Part
719                 tempr = data[n];
720                 data[n] = data[m];
721                 data[m] = tempr;
722                 // Imagery Part
723                 n++;
724                 m++;
725                 tempi = data[n];
726                 data[n] = data[m];
727                 data[m] = tempi;
728             }
729
730             // Performing Bit-Reversal, odd v.s. even, O(N)
731
732             int m = j + fftFrameSize; // bitm_array[i+2];
733             // COMPLEX: SWAP(data[n], data[m])
734             // Real Part
735             int n = i + 2;
736             double tempr = data[n];
737             data[n] = data[m];
738             data[m] = tempr;
739             // Imagery Part
740             n++;
741             m++;
742             double tempi = data[n];
```

```
743                 data[n] = data[m];
744                 data[m] = tempi;
745             }
746 
747         }
748 }
```

33 com/sun/media/sound/InvalidDataException.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.io.IOException;

/**
 * This exception is used when a file contains illegal or unexpected data.
 *
 * @author Karl Helgason
 */
public class InvalidDataException extends IOException {

    private static final long serialVersionUID = 1L;

    public InvalidDataException() {
        super("Invalid Data!");
    }

    public InvalidDataException(String s) {
        super(s);
    }
}
```

34 com/sun/media/sound/InvalidFormatException.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * This exception is used when a reader is used to read file of a format
 * it doesn't unterstand or support.
 *
 * @author Karl Helgason
 */
public class InvalidFormatException extends InvalidDataException {

    private static final long serialVersionUID = 1L;

    public InvalidFormatException() {
        super("Invalid format!");
    }

    public InvalidFormatException(String s) {
        super(s);
    }
}
```

35 com/sun/media/sound/JARSoundbankReader.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.URL;
import java.net.URLClassLoader;
import java.util.ArrayList;
import javax.sound.midi.InvalidMidiDataException;
import javax.sound.midi.Soundbank;
import javax.sound.midi.spi.SoundbankReader;

/**
 * JarSoundbankReader is used to read sounbank object from jar files.
 *
 * @author Karl Helgason
 */
public class JARSoundbankReader extends SoundbankReader {

    public boolean isZIP(URL url) {
        boolean ok = false;
        try {
            InputStream stream = url.openStream();
            try {
                byte[] buff = new byte[4];
                ok = stream.read(buff) == 4;
                if (ok) {
                    ok =  (buff[0] == 0x50
                        && buff[1] == 0x4b
                        && buff[2] == 0x03
                        && buff[3] == 0x04);
                }
            } finally {
                stream.close();
```

```
            }
        } catch (IOException e) {
        }
        return ok;
    }

    public Soundbank getSoundbank(URL url)
            throws InvalidMidiDataException, IOException {
        if (!isZIP(url))
            return null;
        ArrayList<Soundbank> soundbanks = new ArrayList<Soundbank>();
        URLClassLoader ucl = URLClassLoader.newInstance(new URL[]{url});
        InputStream stream = ucl.getResourceAsStream(
                "META-INF/services/javax.sound.midi.Soundbank");
        if (stream == null)
            return null;
        try
        {
            BufferedReader r = new BufferedReader(new InputStreamReader(stream));
            String line = r.readLine();
            while (line != null) {
                if (!line.startsWith("#")) {
                    try {
                        Class c = Class.forName(line.trim(), true, ucl);
                        Object o = c.newInstance();
                        if (o instanceof Soundbank) {
                            soundbanks.add((Soundbank) o);
                        }
                    } catch (ClassNotFoundException  e) {
                    } catch (InstantiationException  e) {
                    } catch (IllegalAccessException  e) {
                    }
                }
                line = r.readLine();
            }
        }
        finally
        {
            stream.close();
        }
        if (soundbanks.size() == 0)
            return null;
        if (soundbanks.size() == 1)
            return soundbanks.get(0);
        SimpleSoundbank sbk = new SimpleSoundbank();
        for (Soundbank soundbank : soundbanks)
            sbk.addAllInstruments(soundbank);
        return sbk;
    }

    public Soundbank getSoundbank(InputStream stream)
            throws InvalidMidiDataException, IOException {
        return null;
    }

    public Soundbank getSoundbank(File file)
            throws InvalidMidiDataException, IOException {
        return getSoundbank(file.toURI().toURL());
    }
}
```

36 com/sun/media/sound/MidiDeviceReceiver.java

```java
/*
 * Copyright 2009 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import javax.sound.midi.MidiDevice;
import javax.sound.midi.Receiver;

/**
 * A Receiver with reference to it's MidiDevice object.
 *
 * @author Karl Helgason
 */
public interface MidiDeviceReceiver extends Receiver {

    /** Obtains the MidiDevice object associated with this Receiver.
     */
    public MidiDevice getMidiDevice();

}
```

37 com/sun/media/sound/ModelAbstractChannelMixer.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * ModelAbstractChannelMixer is ready for use class to implement
 * ModelChannelMixer interface.
 *
 * @author Karl Helgason
 */
public abstract class ModelAbstractChannelMixer implements ModelChannelMixer {

    public abstract boolean process(float[][] buffer, int offset, int len);

    public abstract void stop();

    public void allNotesOff() {
    }

    public void allSoundOff() {
    }

    public void controlChange(int controller, int value) {
    }

    public int getChannelPressure() {
        return 0;
    }

    public int getController(int controller) {
        return 0;
    }

    public boolean getMono() {
        return false;
    }

    public boolean getMute() {
```

```
61          return false;
62      }
63
64      public boolean getOmni() {
65          return false;
66      }
67
68      public int getPitchBend() {
69          return 0;
70      }
71
72      public int getPolyPressure(int noteNumber) {
73          return 0;
74      }
75
76      public int getProgram() {
77          return 0;
78      }
79
80      public boolean getSolo() {
81          return false;
82      }
83
84      public boolean localControl(boolean on) {
85          return false;
86      }
87
88      public void noteOff(int noteNumber) {
89      }
90
91      public void noteOff(int noteNumber, int velocity) {
92      }
93
94      public void noteOn(int noteNumber, int velocity) {
95      }
96
97      public void programChange(int program) {
98      }
99
100     public void programChange(int bank, int program) {
101     }
102
103     public void resetAllControllers() {
104     }
105
106     public void setChannelPressure(int pressure) {
107     }
108
109     public void setMono(boolean on) {
110     }
111
112     public void setMute(boolean mute) {
113     }
114
115     public void setOmni(boolean on) {
116     }
117
118     public void setPitchBend(int bend) {
119     }
120
121     public void setPolyPressure(int noteNumber, int pressure) {
122     }
```

```
123
124     public void setSolo(boolean soloState) {
125     }
126 }
```

38 com/sun/media/sound/ModelAbstractOscillator.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.io.IOException;
import javax.sound.midi.Instrument;
import javax.sound.midi.MidiChannel;
import javax.sound.midi.Patch;
import javax.sound.midi.Soundbank;
import javax.sound.midi.SoundbankResource;
import javax.sound.midi.VoiceStatus;

/**
 * A abstract class used to simplify creating custom ModelOscillator.
 *
 * @author Karl Helgason
 */
public abstract class ModelAbstractOscillator
        implements ModelOscillator, ModelOscillatorStream, Soundbank {

    protected float pitch = 6000;
    protected float samplerate;
    protected MidiChannel channel;
    protected VoiceStatus voice;
    protected int noteNumber;
    protected int velocity;
    protected boolean on = false;

    public void init() {
    }

    public void close() throws IOException {
    }

    public void noteOff(int velocity) {
        on = false;
    }

```

```
61      public void noteOn(MidiChannel channel, VoiceStatus voice, int noteNumber,
62              int velocity) {
63          this.channel = channel;
64          this.voice = voice;
65          this.noteNumber = noteNumber;
66          this.velocity = velocity;
67          on = true;
68      }
69
70      public int read(float[][] buffer, int offset, int len) throws IOException {
71          return -1;
72      }
73
74      public MidiChannel getChannel() {
75          return channel;
76      }
77
78      public VoiceStatus getVoice() {
79          return voice;
80      }
81
82      public int getNoteNumber() {
83          return noteNumber;
84      }
85
86      public int getVelocity() {
87          return velocity;
88      }
89
90      public boolean isOn() {
91          return on;
92      }
93
94      public void setPitch(float pitch) {
95          this.pitch = pitch;
96      }
97
98      public float getPitch() {
99          return pitch;
100     }
101
102     public void setSampleRate(float samplerate) {
103         this.samplerate = samplerate;
104     }
105
106     public float getSampleRate() {
107         return samplerate;
108     }
109
110     public float getAttenuation() {
111         return 0;
112     }
113
114     public int getChannels() {
115         return 1;
116     }
117
118     public String getName() {
119         return getClass().getName();
120     }
121
122     public Patch getPatch() {
```

```
123         return new Patch(0, 0);
124     }
125 
126     public ModelOscillatorStream open(float samplerate) {
127         ModelAbstractOscillator oscs;
128         try {
129             oscs = this.getClass().newInstance();
130         } catch (InstantiationException e) {
131             throw new IllegalArgumentException(e);
132         } catch (IllegalAccessException e) {
133             throw new IllegalArgumentException(e);
134         }
135         oscs.setSampleRate(samplerate);
136         oscs.init();
137         return oscs;
138     }
139 
140     public ModelPerformer getPerformer() {
141         // Create performer for my custom oscillirator
142         ModelPerformer performer = new ModelPerformer();
143         performer.getOscillators().add(this);
144         return performer;
145 
146     }
147 
148     public ModelInstrument getInstrument() {
149         // Create Instrument object around my performer
150         SimpleInstrument ins = new SimpleInstrument();
151         ins.setName(getName());
152         ins.add(getPerformer());
153         ins.setPatch(getPatch());
154         return ins;
155 
156     }
157 
158     public Soundbank getSoundBank() {
159         // Create Soundbank object around the instrument
160         SimpleSoundbank sbk = new SimpleSoundbank();
161         sbk.addInstrument(getInstrument());
162         return sbk;
163     }
164 
165     public String getDescription() {
166         return getName();
167     }
168 
169     public Instrument getInstrument(Patch patch) {
170         Instrument ins = getInstrument();
171         Patch p = ins.getPatch();
172         if (p.getBank() != patch.getBank())
173             return null;
174         if (p.getProgram() != patch.getProgram())
175             return null;
176         if (p instanceof ModelPatch && patch instanceof ModelPatch) {
177             if (((ModelPatch)p).isPercussion()
178                     != ((ModelPatch)patch).isPercussion()) {
179                 return null;
180             }
181         }
182         return ins;
183     }
184 
```

```
185     public Instrument[] getInstruments() {
186         return new Instrument[]{getInstrument()};
187     }
188 
189     public SoundbankResource[] getResources() {
190         return new SoundbankResource[0];
191     }
192 
193     public String getVendor() {
194         return null;
195     }
196 
197     public String getVersion() {
198         return null;
199     }
200 }
```

39 com/sun/media/sound/ModelByteBuffer.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.io.ByteArrayInputStream;
import java.io.DataInputStream;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.RandomAccessFile;
import java.util.Collection;

/**
 * This class is a pointer to a binary array either in memory or on disk.
 *
 * @author Karl Helgason
 */
public class ModelByteBuffer {

    private ModelByteBuffer root = this;
    private File file;
    private long fileoffset;
    private byte[] buffer;
    private long offset;
    private final long len;

    private class RandomFileInputStream extends InputStream {

        private RandomAccessFile raf;
        private long left;
        private long mark = 0;
        private long markleft = 0;

        public RandomFileInputStream() throws IOException {
            raf = new RandomAccessFile(root.file, "r");
            raf.seek(root.fileoffset + arrayOffset());
            left = capacity();
```

```
61          }
62
63          public int available() throws IOException {
64              if (left > Integer.MAX_VALUE)
65                  return Integer.MAX_VALUE;
66              return (int)left;
67          }
68
69          public synchronized void mark(int readlimit) {
70              try {
71                  mark = raf.getFilePointer();
72                  markleft = left;
73              } catch (IOException e) {
74                  //e.printStackTrace();
75              }
76          }
77
78          public boolean markSupported() {
79              return true;
80          }
81
82          public synchronized void reset() throws IOException {
83              raf.seek(mark);
84              left = markleft;
85          }
86
87          public long skip(long n) throws IOException {
88              if( n < 0)
89                  return 0;
90              if (n > left)
91                  n = left;
92              long p = raf.getFilePointer();
93              raf.seek(p + n);
94              left -= n;
95              return n;
96          }
97
98          public int read(byte b[], int off, int len) throws IOException {
99              if (len > left)
100                 len = (int)left;
101             if (left == 0)
102                 return -1;
103             len = raf.read(b, off, len);
104             if (len == -1)
105                 return -1;
106             left -= len;
107             return len;
108         }
109
110         public int read(byte[] b) throws IOException {
111             int len = b.length;
112             if (len > left)
113                 len = (int)left;
114             if (left == 0)
115                 return -1;
116             len = raf.read(b, 0, len);
117             if (len == -1)
118                 return -1;
119             left -= len;
120             return len;
121         }
122
```

```
        public int read() throws IOException {
            if (left == 0)
                return -1;
            int b = raf.read();
            if (b == -1)
                return -1;
            left--;
            return b;
        }

        public void close() throws IOException {
            raf.close();
        }
    }

    private ModelByteBuffer(ModelByteBuffer parent,
            long beginIndex, long endIndex, boolean independent) {
        this.root = parent.root;
        this.offset = 0;
        long parent_len = parent.len;
        if (beginIndex < 0)
            beginIndex = 0;
        if (beginIndex > parent_len)
            beginIndex = parent_len;
        if (endIndex < 0)
            endIndex = 0;
        if (endIndex > parent_len)
            endIndex = parent_len;
        if (beginIndex > endIndex)
            beginIndex = endIndex;
        offset = beginIndex;
        len = endIndex - beginIndex;
        if (independent) {
            buffer = root.buffer;
            if (root.file != null) {
                file = root.file;
                fileoffset = root.fileoffset + arrayOffset();
                offset = 0;
            } else
                offset = arrayOffset();
            root = this;
        }
    }

    public ModelByteBuffer(byte[] buffer) {
        this.buffer = buffer;
        this.offset = 0;
        this.len = buffer.length;
    }

    public ModelByteBuffer(byte[] buffer, int offset, int len) {
        this.buffer = buffer;
        this.offset = offset;
        this.len = len;
    }

    public ModelByteBuffer(File file) {
        this.file = file;
        this.fileoffset = 0;
        this.len = file.length();
    }

```

```
    public ModelByteBuffer(File file, long offset, long len) {
        this.file = file;
        this.fileoffset = offset;
        this.len = len;
    }

    public void writeTo(OutputStream out) throws IOException {
        if (root.file != null && root.buffer == null) {
            InputStream is = getInputStream();
            byte[] buff = new byte[1024];
            int ret;
            while ((ret = is.read(buff)) != -1)
                out.write(buff, 0, ret);
        } else
            out.write(array(), (int) arrayOffset(), (int) capacity());
    }

    public InputStream getInputStream() {
        if (root.file != null && root.buffer == null) {
            try {
                return new RandomFileInputStream();
            } catch (IOException e) {
                //e.printStackTrace();
                return null;
            }
        }
        return new ByteArrayInputStream(array(),
                (int)arrayOffset(), (int)capacity());
    }

    public ModelByteBuffer subbuffer(long beginIndex) {
        return subbuffer(beginIndex, capacity());
    }

    public ModelByteBuffer subbuffer(long beginIndex, long endIndex) {
        return subbuffer(beginIndex, endIndex, false);
    }

    public ModelByteBuffer subbuffer(long beginIndex, long endIndex,
            boolean independent) {
        return new ModelByteBuffer(this, beginIndex, endIndex, independent);
    }

    public byte[] array() {
        return root.buffer;
    }

    public long arrayOffset() {
        if (root != this)
            return root.arrayOffset() + offset;
        return offset;
    }

    public long capacity() {
        return len;
    }

    public ModelByteBuffer getRoot() {
        return root;
    }

    public File getFile() {
```

```
247         return file;
248     }
249 
250     public long getFilePointer() {
251         return fileoffset;
252     }
253 
254     public static void loadAll(Collection<ModelByteBuffer> col)
255             throws IOException {
256         File selfile = null;
257         RandomAccessFile raf = null;
258         try {
259             for (ModelByteBuffer mbuff : col) {
260                 mbuff = mbuff.root;
261                 if (mbuff.file == null)
262                     continue;
263                 if (mbuff.buffer != null)
264                     continue;
265                 if (selfile == null || !selfile.equals(mbuff.file)) {
266                     if (raf != null) {
267                         raf.close();
268                         raf = null;
269                     }
270                     selfile = mbuff.file;
271                     raf = new RandomAccessFile(mbuff.file, "r");
272                 }
273                 raf.seek(mbuff.fileoffset);
274                 byte[] buffer = new byte[(int) mbuff.capacity()];
275 
276                 int read = 0;
277                 int avail = buffer.length;
278                 while (read != avail) {
279                     if (avail - read > 65536) {
280                         raf.readFully(buffer, read, 65536);
281                         read += 65536;
282                     } else {
283                         raf.readFully(buffer, read, avail - read);
284                         read = avail;
285                     }
286 
287                 }
288 
289                 mbuff.buffer = buffer;
290                 mbuff.offset = 0;
291             }
292         } finally {
293             if (raf != null)
294                 raf.close();
295         }
296     }
297 
298     public void load() throws IOException {
299         if (root != this) {
300             root.load();
301             return;
302         }
303         if (buffer != null)
304             return;
305         if (file == null) {
306             throw new IllegalStateException(
307                     "No file associated with this ByteBuffer!");
308         }
```

```
309
310         DataInputStream is = new DataInputStream(getInputStream());
311         buffer = new byte[(int) capacity()];
312         offset = 0;
313         is.readFully(buffer);
314         is.close();
315
316     }
317
318     public void unload() {
319         if (root != this) {
320             root.unload();
321             return;
322         }
323         if (file == null) {
324             throw new IllegalStateException(
325                     "No file associated with this ByteBuffer!");
326         }
327         root.buffer = null;
328     }
329 }
```

40 com/sun/media/sound/ModelByteBufferWavetable.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.io.IOException;
import java.io.InputStream;
import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.AudioFormat.Encoding;

/**
 * Wavetable oscillator for pre-loaded data.
 *
 * @author Karl Helgason
 */
public class ModelByteBufferWavetable implements ModelWavetable {

    private class Buffer8PlusInputStream extends InputStream {

        private boolean bigendian;
        private int framesize_pc;
        int pos = 0;
        int pos2 = 0;
        int markpos = 0;
        int markpos2 = 0;

        public Buffer8PlusInputStream() {
            framesize_pc = format.getFrameSize() / format.getChannels();
            bigendian = format.isBigEndian();
        }

        public int read(byte[] b, int off, int len) throws IOException {
            int avail = available();
            if (avail <= 0)
                return -1;
            if (len > avail)
                len = avail;
```

```
            byte[] buff1 = buffer.array();
            byte[] buff2 = buffer8.array();
            pos += buffer.arrayOffset();
            pos2 += buffer8.arrayOffset();
            if (bigendian) {
                for (int i = 0; i < len; i += (framesize_pc + 1)) {
                    System.arraycopy(buff1, pos, b, i, framesize_pc);
                    System.arraycopy(buff2, pos2, b, i + framesize_pc, 1);
                    pos += framesize_pc;
                    pos2 += 1;
                }
            } else {
                for (int i = 0; i < len; i += (framesize_pc + 1)) {
                    System.arraycopy(buff2, pos2, b, i, 1);
                    System.arraycopy(buff1, pos, b, i + 1, framesize_pc);
                    pos += framesize_pc;
                    pos2 += 1;
                }
            }
            pos -= buffer.arrayOffset();
            pos2 -= buffer8.arrayOffset();
            return len;
        }

        public long skip(long n) throws IOException {
            int avail = available();
            if (avail <= 0)
                return -1;
            if (n > avail)
                n = avail;
            pos += (n / (framesize_pc + 1)) * (framesize_pc);
            pos2 += n / (framesize_pc + 1);
            return super.skip(n);
        }

        public int read(byte[] b) throws IOException {
            return read(b, 0, b.length);
        }

        public int read() throws IOException {
            byte[] b = new byte[1];
            int ret = read(b, 0, 1);
            if (ret == -1)
                return -1;
            return 0 & 0xFF;
        }

        public boolean markSupported() {
            return true;
        }

        public int available() throws IOException {
            return (int)buffer.capacity() + (int)buffer8.capacity() - pos - pos2;
        }

        public synchronized void mark(int readlimit) {
            markpos = pos;
            markpos2 = pos2;
        }

        public synchronized void reset() throws IOException {
            pos = markpos;
```

```
            pos2 = markpos2;

        }
    }

    private float loopStart = -1;
    private float loopLength = -1;
    private ModelByteBuffer buffer;
    private ModelByteBuffer buffer8 = null;
    private AudioFormat format = null;
    private float pitchcorrection = 0;
    private float attenuation = 0;
    private int loopType = LOOP_TYPE_OFF;

    public ModelByteBufferWavetable(ModelByteBuffer buffer) {
        this.buffer = buffer;
    }

    public ModelByteBufferWavetable(ModelByteBuffer buffer,
            float pitchcorrection) {
        this.buffer = buffer;
        this.pitchcorrection = pitchcorrection;
    }

    public ModelByteBufferWavetable(ModelByteBuffer buffer, AudioFormat format) {
        this.format = format;
        this.buffer = buffer;
    }

    public ModelByteBufferWavetable(ModelByteBuffer buffer, AudioFormat format,
            float pitchcorrection) {
        this.format = format;
        this.buffer = buffer;
        this.pitchcorrection = pitchcorrection;
    }

    public void set8BitExtensionBuffer(ModelByteBuffer buffer) {
        buffer8 = buffer;
    }

    public ModelByteBuffer get8BitExtensionBuffer() {
        return buffer8;
    }

    public ModelByteBuffer getBuffer() {
        return buffer;
    }

    public AudioFormat getFormat() {
        if (format == null) {
            if (buffer == null)
                return null;
            InputStream is = buffer.getInputStream();
            AudioFormat format = null;
            try {
                format = AudioSystem.getAudioFileFormat(is).getFormat();
            } catch (Exception e) {
                //e.printStackTrace();
            }
            try {
                is.close();
            } catch (IOException e) {
```

```
                //e.printStackTrace();
            }
            return format;
        }
        return format;
    }

    public AudioFloatInputStream openStream() {
        if (buffer == null)
            return null;
        if (format == null) {
            InputStream is = buffer.getInputStream();
            AudioInputStream ais = null;
            try {
                ais = AudioSystem.getAudioInputStream(is);
            } catch (Exception e) {
                //e.printStackTrace();
                return null;
            }
            return AudioFloatInputStream.getInputStream(ais);
        }
        if (buffer.array() == null) {
            return AudioFloatInputStream.getInputStream(new AudioInputStream(
                    buffer.getInputStream(), format,
                    buffer.capacity() / format.getFrameSize()));
        }
        if (buffer8 != null) {
            if (format.getEncoding().equals(Encoding.PCM_SIGNED)
                    || format.getEncoding().equals(Encoding.PCM_UNSIGNED)) {
                InputStream is = new Buffer8PlusInputStream();
                AudioFormat format2 = new AudioFormat(
                        format.getEncoding(),
                        format.getSampleRate(),
                        format.getSampleSizeInBits() + 8,
                        format.getChannels(),
                        format.getFrameSize() + (1 * format.getChannels()),
                        format.getFrameRate(),
                        format.isBigEndian());

                AudioInputStream ais = new AudioInputStream(is, format2,
                        buffer.capacity() / format.getFrameSize());
                return AudioFloatInputStream.getInputStream(ais);
            }
        }
        return AudioFloatInputStream.getInputStream(format, buffer.array(),
                (int)buffer.arrayOffset(), (int)buffer.capacity());
    }

    public int getChannels() {
        return getFormat().getChannels();
    }

    public ModelOscillatorStream open(float samplerate) {
        // ModelWavetableOscillator doesn't support ModelOscillatorStream
        return null;
    }

    // attenuation is in cB
    public float getAttenuation() {
        return attenuation;
    }
    // attenuation is in cB
```

```
    public void setAttenuation(float attenuation) {
        this.attenuation = attenuation;
    }

    public float getLoopLength() {
        return loopLength;
    }

    public void setLoopLength(float loopLength) {
        this.loopLength = loopLength;
    }

    public float getLoopStart() {
        return loopStart;
    }

    public void setLoopStart(float loopStart) {
        this.loopStart = loopStart;
    }

    public void setLoopType(int loopType) {
        this.loopType = loopType;
    }

    public int getLoopType() {
        return loopType;
    }

    public float getPitchcorrection() {
        return pitchcorrection;
    }

    public void setPitchcorrection(float pitchcorrection) {
        this.pitchcorrection = pitchcorrection;
    }
}
```

41 com/sun/media/sound/ModelChannelMixer.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import javax.sound.midi.MidiChannel;

/**
 * ModelChannelMixer is used to process channel voice mix output before going
 * to master output.<br>
 * It can be used to:<br>
 * <ul>
 *   <li>Implement non-voice oriented instruments.</li>
 *   <li>Add insert effect to instruments; for example distortion effect.</li>
 * </ui>
 * <p>
 * <b>Warning! Classes that implements ModelChannelMixer must be thread-safe.</b>
 *
 * @author Karl Helgason
 */
public interface ModelChannelMixer extends MidiChannel {

    // Used to process input audio from voices mix.
    public boolean process(float[][] buffer, int offset, int len);

    // Is used to trigger that this mixer is not be used
    // and it should fade out.
    public void stop();
}
```

42 com/sun/media/sound/ModelConnectionBlock.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * Connection blocks are used to connect source variable
 * to a destination variable.
 * For example Note On velocity can be connected to output gain.
 * In DLS this is called articulator and in SoundFonts (SF2) a modulator.
 *
 * @author Karl Helgason
 */
public class ModelConnectionBlock {

    //
    //   source1 * source2 * scale -> destination
    //
    private final static ModelSource[] no_sources = new ModelSource[0];
    private ModelSource[] sources = no_sources;
    private double scale = 1;
    private ModelDestination destination;

    public ModelConnectionBlock() {
    }

    public ModelConnectionBlock(double scale, ModelDestination destination) {
        this.scale = scale;
        this.destination = destination;
    }

    public ModelConnectionBlock(ModelSource source,
            ModelDestination destination) {
        if (source != null) {
            this.sources = new ModelSource[1];
            this.sources[0] = source;
        }
        this.destination = destination;
    }
```

```
61
62      public ModelConnectionBlock(ModelSource source, double scale,
63              ModelDestination destination) {
64          if (source != null) {
65              this.sources = new ModelSource[1];
66              this.sources[0] = source;
67          }
68          this.scale = scale;
69          this.destination = destination;
70      }
71
72      public ModelConnectionBlock(ModelSource source, ModelSource control,
73              ModelDestination destination) {
74          if (source != null) {
75              if (control == null) {
76                  this.sources = new ModelSource[1];
77                  this.sources[0] = source;
78              } else {
79                  this.sources = new ModelSource[2];
80                  this.sources[0] = source;
81                  this.sources[1] = control;
82              }
83          }
84          this.destination = destination;
85      }
86
87      public ModelConnectionBlock(ModelSource source, ModelSource control,
88              double scale, ModelDestination destination) {
89          if (source != null) {
90              if (control == null) {
91                  this.sources = new ModelSource[1];
92                  this.sources[0] = source;
93              } else {
94                  this.sources = new ModelSource[2];
95                  this.sources[0] = source;
96                  this.sources[1] = control;
97              }
98          }
99          this.scale = scale;
100         this.destination = destination;
101     }
102
103     public ModelDestination getDestination() {
104         return destination;
105     }
106
107     public void setDestination(ModelDestination destination) {
108         this.destination = destination;
109     }
110
111     public double getScale() {
112         return scale;
113     }
114
115     public void setScale(double scale) {
116         this.scale = scale;
117     }
118
119     public ModelSource[] getSources() {
120         return sources;
121     }
122
```

```
123     public void setSources(ModelSource[] source) {
124         this.sources = source;
125     }
126 
127     public void addSource(ModelSource source) {
128         ModelSource[] oldsources = sources;
129         sources = new ModelSource[oldsources.length + 1];
130         for (int i = 0; i < oldsources.length; i++) {
131             sources[i] = oldsources[i];
132         }
133         sources[sources.length - 1] = source;
134     }
135 }
```

43 com/sun/media/sound/ModelDestination.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * This class is used to identify destinations in connection blocks,
 * see ModelConnectionBlock.
 *
 * @author Karl Helgason
 */
public class ModelDestination {

    public static final ModelIdentifier DESTINATION_NONE = null;
    public static final ModelIdentifier DESTINATION_KEYNUMBER
            = new ModelIdentifier("noteon", "keynumber");
    public static final ModelIdentifier DESTINATION_VELOCITY
            = new ModelIdentifier("noteon", "velocity");
    public static final ModelIdentifier DESTINATION_PITCH
            = new ModelIdentifier("osc", "pitch");   // cent
    public static final ModelIdentifier DESTINATION_GAIN
            = new ModelIdentifier("mixer", "gain");   // cB
    public static final ModelIdentifier DESTINATION_PAN
            = new ModelIdentifier("mixer", "pan");   // 0.1 %
    public static final ModelIdentifier DESTINATION_REVERB
            = new ModelIdentifier("mixer", "reverb");   // 0.1 %
    public static final ModelIdentifier DESTINATION_CHORUS
            = new ModelIdentifier("mixer", "chorus");   // 0.1 %
    public static final ModelIdentifier DESTINATION_LFO1_DELAY
            = new ModelIdentifier("lfo", "delay", 0); // timecent
    public static final ModelIdentifier DESTINATION_LFO1_FREQ
            = new ModelIdentifier("lfo", "freq", 0); // cent
    public static final ModelIdentifier DESTINATION_LFO2_DELAY
            = new ModelIdentifier("lfo", "delay", 1); // timecent
    public static final ModelIdentifier DESTINATION_LFO2_FREQ
            = new ModelIdentifier("lfo", "freq", 1); // cent
    public static final ModelIdentifier DESTINATION_EG1_DELAY
            = new ModelIdentifier("eg", "delay", 0); // timecent
    public static final ModelIdentifier DESTINATION_EG1_ATTACK
```

```
            = new ModelIdentifier("eg", "attack", 0); // timecent
    public static final ModelIdentifier DESTINATION_EG1_HOLD
            = new ModelIdentifier("eg", "hold", 0); // timecent
    public static final ModelIdentifier DESTINATION_EG1_DECAY
            = new ModelIdentifier("eg", "decay", 0); // timecent
    public static final ModelIdentifier DESTINATION_EG1_SUSTAIN
            = new ModelIdentifier("eg", "sustain", 0);
                                       // 0.1 % (I want this to be value not %)
    public static final ModelIdentifier DESTINATION_EG1_RELEASE
            = new ModelIdentifier("eg", "release", 0); // timecent
    public static final ModelIdentifier DESTINATION_EG1_SHUTDOWN
            = new ModelIdentifier("eg", "shutdown", 0); // timecent
    public static final ModelIdentifier DESTINATION_EG2_DELAY
            = new ModelIdentifier("eg", "delay", 1); // timecent
    public static final ModelIdentifier DESTINATION_EG2_ATTACK
            = new ModelIdentifier("eg", "attack", 1); // timecent
    public static final ModelIdentifier DESTINATION_EG2_HOLD
            = new ModelIdentifier("eg", "hold", 1); // 0.1 %
    public static final ModelIdentifier DESTINATION_EG2_DECAY
            = new ModelIdentifier("eg", "decay", 1); // timecent
    public static final ModelIdentifier DESTINATION_EG2_SUSTAIN
            = new ModelIdentifier("eg", "sustain", 1);
                                       // 0.1 % ( I want this to be value not %)
    public static final ModelIdentifier DESTINATION_EG2_RELEASE
            = new ModelIdentifier("eg", "release", 1); // timecent
    public static final ModelIdentifier DESTINATION_EG2_SHUTDOWN
            = new ModelIdentifier("eg", "shutdown", 1); // timecent
    public static final ModelIdentifier DESTINATION_FILTER_FREQ
            = new ModelIdentifier("filter", "freq", 0); // cent
    public static final ModelIdentifier DESTINATION_FILTER_Q
            = new ModelIdentifier("filter", "q", 0); // cB
    private ModelIdentifier destination = DESTINATION_NONE;
    private ModelTransform transform = new ModelStandardTransform();

    public ModelDestination() {
    }

    public ModelDestination(ModelIdentifier id) {
        destination = id;
    }

    public ModelIdentifier getIdentifier() {
        return destination;
    }

    public void setIdentifier(ModelIdentifier destination) {
        this.destination = destination;
    }

    public ModelTransform getTransform() {
        return transform;
    }

    public void setTransform(ModelTransform transform) {
        this.transform = transform;
    }
}
```

44 com/sun/media/sound/ModelDirectedPlayer.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 *  ModelDirectedPlayer is the one who is directed by ModelDirector
 *  to play ModelPerformer objects.
 *
 * @author Karl Helgason
 */
public interface ModelDirectedPlayer {

    public void play(int performerIndex, ModelConnectionBlock[] connectionBlocks);
}
```

45 com/sun/media/sound/ModelDirector.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * A director chooses what performers should be  played for each note on
 * and note off events.
 *
 * ModelInstrument can implement custom performer who chooses what performers
 * to play for example by sustain pedal is off or on.
 *
 * The default director (ModelStandardDirector) chooses performers
 * by there keyfrom,keyto,velfrom,velto properties.
 *
 * @author Karl Helgason
 */
public interface ModelDirector {

    public void noteOn(int noteNumber, int velocity);

    public void noteOff(int noteNumber, int velocity);

    public void close();
}
```

46 com/sun/media/sound/ModelIdentifier.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * This class stores the identity of source and destinations in connection
 * blocks, see ModelConnectionBlock.
 *
 * @author Karl Helgason
 */
public class ModelIdentifier {

    /*
     *  Object    Variable
     *  ------    --------
     *
     *  // INPUT parameters
     *  noteon    keynumber               7 bit midi value
     *            velocity                7 bit midi vale
     *            on                      1 or 0
     *
     *  midi      pitch                   14 bit midi value
     *            channel_pressure        7 bit midi value
     *            poly_pressure           7 bit midi value
     *
     *  midi_cc   0 (midi control #0      7 bit midi value
     *            1 (midi control #1      7 bit midi value
     *            ...
     *            127 (midi control #127  7 bit midi value
     *
     *  midi_rpn  0 (midi rpn control #0) 14 bit midi value
     *            1 (midi rpn control #1) 14 bit midi value
     *            ....
     *
     *  // DAHDSR envelope generator
     *  eg        (null)
     *            delay                   timecent
     *            attack                  timecent
```

```
     *            hold                    timecent
     *            decay                   timecent
     *            sustain                 0.1 %
     *            release                 timecent
     *
     *  // Low frequency oscillirator (sine wave)
     *  lfo       (null)
     *            delay                   timcent
     *            freq                    cent
     *
     *  // Resonance LowPass Filter 6dB slope
     *  filter    (null) (output/input)
     *            freq                    cent
     *            q                       cB
     *
     *  // The oscillator with preloaded wavetable data
     *  osc       (null)
     *            pitch                   cent
     *
     *  // Output mixer pins
     *  mixer     gain                    cB
     *            pan                     0.1 %
     *            reverb                  0.1 %
     *            chorus                  0.1 %
     *
     */
    private String object = null;
    private String variable = null;
    private int instance = 0;

    public ModelIdentifier(String object) {
        this.object = object;
    }

    public ModelIdentifier(String object, int instance) {
        this.object = object;
        this.instance = instance;
    }

    public ModelIdentifier(String object, String variable) {
        this.object = object;
        this.variable = variable;

    }

    public ModelIdentifier(String object, String variable, int instance) {
        this.object = object;
        this.variable = variable;
        this.instance = instance;

    }

    public int getInstance() {
        return instance;
    }

    public void setInstance(int instance) {
        this.instance = instance;
    }

    public String getObject() {
        return object;
```

```
    }

    public void setObject(String object) {
        this.object = object;
    }

    public String getVariable() {
        return variable;
    }

    public void setVariable(String variable) {
        this.variable = variable;
    }

  public int hashCode() {
    int hashcode = instance;
    if(object != null) hashcode |= object.hashCode();
    if(variable != null) hashcode |= variable.hashCode();
    return  hashcode;
  }

    public boolean equals(Object obj) {
        if (!(obj instanceof ModelIdentifier))
            return false;

        ModelIdentifier mobj = (ModelIdentifier)obj;
        if ((object == null) != (mobj.object == null))
            return false;
        if ((variable == null) != (mobj.variable == null))
            return false;
        if (mobj.getInstance() != getInstance())
            return false;
        if (!(object == null || object.equals(mobj.object)))
            return false;
        if (!(variable == null || variable.equals(mobj.variable)))
            return false;
        return true;
    }

    public String toString() {
        if (variable == null) {
            return object + "[" + instance + "]";
        } else {
            return object + "[" + instance + "]" + "." + variable;
        }
    }
}
```

47 com/sun/media/sound/ModelInstrument.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import javax.sound.midi.Instrument;
import javax.sound.midi.MidiChannel;
import javax.sound.midi.Patch;
import javax.sound.midi.Soundbank;
import javax.sound.sampled.AudioFormat;

/**
 * The model instrument class.
 *
 * <p>The main methods to override are:<br>
 * getPerformer, getDirector, getChannelMixer.
 *
 * <p>Performers are used to define what voices which will
 * playback when using the instrument.<br>
 *
 * ChannelMixer is used to add channel-wide processing
 * on voices output or to define non-voice oriented instruments.<br>
 *
 * Director is used to change how the synthesizer
 * chooses what performers to play on midi events.
 *
 * @author Karl Helgason
 */
public abstract class ModelInstrument extends Instrument {

    protected ModelInstrument(Soundbank soundbank, Patch patch, String name,
            Class<?> dataClass) {
        super(soundbank, patch, name, dataClass);
    }

    public ModelDirector getDirector(ModelPerformer[] performers,
            MidiChannel channel, ModelDirectedPlayer player) {
        return new ModelStandardIndexedDirector(performers, player);
    }
```

```
61
62      public ModelPerformer[] getPerformers() {
63          return new ModelPerformer[0];
64      }
65
66      public ModelChannelMixer getChannelMixer(MidiChannel channel,
67              AudioFormat format) {
68          return null;
69      }
70
71      // Get General MIDI 2 Alias patch for this instrument.
72      public Patch getPatchAlias() {
73          Patch patch = getPatch();
74          int program = patch.getProgram();
75          int bank = patch.getBank();
76          if (bank != 0)
77              return patch;
78          boolean percussion = false;
79          if (getPatch() instanceof ModelPatch)
80              percussion = ((ModelPatch)getPatch()).isPercussion();
81          if (percussion)
82              return new Patch(0x78 << 7, program);
83          else
84              return new Patch(0x79 << 7, program);
85      }
86
87      // Return name of all the keys.
88      // This information is generated from ModelPerformer.getName()
89      // returned from getPerformers().
90      public String[] getKeys() {
91          String[] keys = new String[128];
92          for (ModelPerformer performer : getPerformers()) {
93              for (int k = performer.getKeyFrom(); k <= performer.getKeyTo(); k++) {
94                  if (k >= 0 && k < 128 && keys[k] == null) {
95                      String name = performer.getName();
96                      if (name == null)
97                          name = "untitled";
98                      keys[k] = name;
99                  }
100             }
101         }
102         return keys;
103     }
104
105     // Return what channels this instrument will probably response
106     // on General MIDI synthesizer.
107     public boolean[] getChannels() {
108         boolean percussion = false;
109         if (getPatch() instanceof ModelPatch)
110             percussion = ((ModelPatch)getPatch()).isPercussion();
111
112         // Check if instrument is percussion.
113         if (percussion) {
114             boolean[] ch = new boolean[16];
115             for (int i = 0; i < ch.length; i++)
116                 ch[i] = false;
117             ch[9] = true;
118             return ch;
119         }
120
121         // Check if instrument uses General MIDI 2 default banks.
122         int bank = getPatch().getBank();
```

```
        if (bank >> 7 == 0x78 || bank >> 7 == 0x79) {
            boolean[] ch = new boolean[16];
            for (int i = 0; i < ch.length; i++)
                ch[i] = true;
            return ch;
        }

        boolean[] ch = new boolean[16];
        for (int i = 0; i < ch.length; i++)
            ch[i] = true;
        ch[9] = false;
        return ch;
    }
}
```

48 com/sun/media/sound/ModelInstrumentComparator.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.util.Comparator;
import javax.sound.midi.Instrument;
import javax.sound.midi.Patch;

/**
 * Instrument comparator class.
 * Used to order instrument by program, bank, percussion.
 *
 * @author Karl Helgason
 */
public class ModelInstrumentComparator implements Comparator<Instrument> {

    public int compare(Instrument arg0, Instrument arg1) {
        Patch p0 = arg0.getPatch();
        Patch p1 = arg1.getPatch();
        int a = p0.getBank() * 128 + p0.getProgram();
        int b = p1.getBank() * 128 + p1.getProgram();
        if (p0 instanceof ModelPatch) {
            a += ((ModelPatch)p0).isPercussion() ? 2097152 : 0;
        }
        if (p1 instanceof ModelPatch) {
            b += ((ModelPatch)p1).isPercussion() ? 2097152 : 0;
        }
        return a - b;
    }
}
```

49 com/sun/media/sound/ModelMappedInstrument.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import javax.sound.midi.MidiChannel;
import javax.sound.midi.Patch;
import javax.sound.sampled.AudioFormat;

/**
 * This class is used to map instrument to another patch.
 *
 * @author Karl Helgason
 */
public class ModelMappedInstrument extends ModelInstrument {

    private ModelInstrument ins;

    public ModelMappedInstrument(ModelInstrument ins, Patch patch) {
        super(ins.getSoundbank(), patch, ins.getName(), ins.getDataClass());
        this.ins = ins;
    }

    public Object getData() {
        return ins.getData();
    }

    public ModelPerformer[] getPerformers() {
        return ins.getPerformers();
    }

    public ModelDirector getDirector(ModelPerformer[] performers,
            MidiChannel channel, ModelDirectedPlayer player) {
        return ins.getDirector(performers, channel, player);
    }

    public ModelChannelMixer getChannelMixer(MidiChannel channel,
            AudioFormat format) {
        return ins.getChannelMixer(channel, format);
```

```
61         }
62 }
```

50 com/sun/media/sound/ModelOscillator.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * This interface is used for oscillators.
 * See example in ModelDefaultOscillator which is a wavetable oscillator.
 *
 * @author Karl Helgason
 */
public interface ModelOscillator {

    public int getChannels();

    /**
     * Attenuation is in cB.
     * @return
     */
    public float getAttenuation();

    public ModelOscillatorStream open(float samplerate);
}
```

51 com/sun/media/sound/ModelOscillatorStream.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.io.IOException;
import javax.sound.midi.MidiChannel;
import javax.sound.midi.VoiceStatus;

/**
 * This interface is used for audio streams from ModelOscillator.
 *
 * @author Karl Helgason
 */
public interface ModelOscillatorStream {

    public void setPitch(float pitch); // Pitch is in cents!

    public void noteOn(MidiChannel channel, VoiceStatus voice, int noteNumber,
            int velocity);

    public void noteOff(int velocity);

    public int read(float[][] buffer, int offset, int len) throws IOException;

    public void close() throws IOException;
}
```

52 com/sun/media/sound/ModelPatch.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import javax.sound.midi.Patch;

/**
 * A extended patch object that has isPercussion function.
 * Which is necessary to identify percussion instruments
 * from melodic instruments.
 *
 * @author Karl Helgason
 */
public class ModelPatch extends Patch {

    private boolean percussion = false;

    public ModelPatch(int bank, int program) {
        super(bank, program);
    }

    public ModelPatch(int bank, int program, boolean percussion) {
        super(bank, program);
        this.percussion = percussion;
    }

    public boolean isPercussion() {
        return percussion;
    }
}
```

53 com/sun/media/sound/ModelPerformer.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

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

/**
 * This class is used to define how to synthesize audio in universal maner
 * for both SF2 and DLS instruments.
 *
 * @author Karl Helgason
 */
public class ModelPerformer {

    private List<ModelOscillator> oscillators = new ArrayList<ModelOscillator>();
    private List<ModelConnectionBlock> connectionBlocks
            = new ArrayList<ModelConnectionBlock>();
    private int keyFrom = 0;
    private int keyTo = 127;
    private int velFrom = 0;
    private int velTo = 127;
    private int exclusiveClass = 0;
    private boolean releaseTrigger = false;
    private boolean selfNonExclusive = false;
    private Object userObject = null;
    private boolean addDefaultConnections = true;
    private String name = null;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public List<ModelConnectionBlock> getConnectionBlocks() {
```

```
        return connectionBlocks;
    }

    public void setConnectionBlocks(List<ModelConnectionBlock> connectionBlocks) {
        this.connectionBlocks = connectionBlocks;
    }

    public List<ModelOscillator> getOscillators() {
        return oscillators;
    }

    public int getExclusiveClass() {
        return exclusiveClass;
    }

    public void setExclusiveClass(int exclusiveClass) {
        this.exclusiveClass = exclusiveClass;
    }

    public boolean isSelfNonExclusive() {
        return selfNonExclusive;
    }

    public void setSelfNonExclusive(boolean selfNonExclusive) {
        this.selfNonExclusive = selfNonExclusive;
    }

    public int getKeyFrom() {
        return keyFrom;
    }

    public void setKeyFrom(int keyFrom) {
        this.keyFrom = keyFrom;
    }

    public int getKeyTo() {
        return keyTo;
    }

    public void setKeyTo(int keyTo) {
        this.keyTo = keyTo;
    }

    public int getVelFrom() {
        return velFrom;
    }

    public void setVelFrom(int velFrom) {
        this.velFrom = velFrom;
    }

    public int getVelTo() {
        return velTo;
    }

    public void setVelTo(int velTo) {
        this.velTo = velTo;
    }

    public boolean isReleaseTriggered() {
        return releaseTrigger;
    }
```

```
123
124     public void setReleaseTriggered(boolean value) {
125         this.releaseTrigger = value;
126     }
127
128     public Object getUserObject() {
129         return userObject;
130     }
131
132     public void setUserObject(Object object) {
133         userObject = object;
134     }
135
136     public boolean isDefaultConnectionsEnabled() {
137         return addDefaultConnections;
138     }
139
140     public void setDefaultConnectionsEnabled(boolean addDefaultConnections) {
141         this.addDefaultConnections = addDefaultConnections;
142     }
143 }
```

54 com/sun/media/sound/ModelSource.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * This class is used to identify sources in connection blocks,
 * see ModelConnectionBlock.
 *
 * @author Karl Helgason
 */
public class ModelSource {

    public static final ModelIdentifier SOURCE_NONE = null;
    public static final ModelIdentifier SOURCE_NOTEON_KEYNUMBER =
            new ModelIdentifier("noteon", "keynumber");     // midi keynumber
    public static final ModelIdentifier SOURCE_NOTEON_VELOCITY =
            new ModelIdentifier("noteon", "velocity");      // midi velocity
    public static final ModelIdentifier SOURCE_EG1 =
            new ModelIdentifier("eg", null, 0);
    public static final ModelIdentifier SOURCE_EG2 =
            new ModelIdentifier("eg", null, 1);
    public static final ModelIdentifier SOURCE_LFO1 =
            new ModelIdentifier("lfo", null, 0);
    public static final ModelIdentifier SOURCE_LFO2 =
            new ModelIdentifier("lfo", null, 1);
    public static final ModelIdentifier SOURCE_MIDI_PITCH =
            new ModelIdentifier("midi", "pitch", 0);            // (0..16383)
    public static final ModelIdentifier SOURCE_MIDI_CHANNEL_PRESSURE =
            new ModelIdentifier("midi", "channel_pressure", 0); // (0..127)
//    public static final ModelIdentifier SOURCE_MIDI_MONO_PRESSURE =
//            new ModelIdentifier("midi","mono_pressure",0);    // (0..127)
    public static final ModelIdentifier SOURCE_MIDI_POLY_PRESSURE =
            new ModelIdentifier("midi", "poly_pressure", 0);    // (0..127)
    public static final ModelIdentifier SOURCE_MIDI_CC_0 =
            new ModelIdentifier("midi_cc", "0", 0);             // (0..127)
    public static final ModelIdentifier SOURCE_MIDI_RPN_0 =
            new ModelIdentifier("midi_rpn", "0", 0);            // (0..16383)
    private ModelIdentifier source = SOURCE_NONE;
```

```
61      private ModelTransform transform;
62
63      public ModelSource() {
64          this.transform = new ModelStandardTransform();
65      }
66
67      public ModelSource(ModelIdentifier id) {
68          source = id;
69          this.transform = new ModelStandardTransform();
70      }
71
72      public ModelSource(ModelIdentifier id, boolean direction) {
73          source = id;
74          this.transform = new ModelStandardTransform(direction);
75      }
76
77      public ModelSource(ModelIdentifier id, boolean direction, boolean polarity) {
78          source = id;
79          this.transform = new ModelStandardTransform(direction, polarity);
80      }
81
82      public ModelSource(ModelIdentifier id, boolean direction, boolean polarity,
83              int transform) {
84          source = id;
85          this.transform =
86                  new ModelStandardTransform(direction, polarity, transform);
87      }
88
89      public ModelSource(ModelIdentifier id, ModelTransform transform) {
90          source = id;
91          this.transform = transform;
92      }
93
94      public ModelIdentifier getIdentifier() {
95          return source;
96      }
97
98      public void setIdentifier(ModelIdentifier source) {
99          this.source = source;
100     }
101
102     public ModelTransform getTransform() {
103         return transform;
104     }
105
106     public void setTransform(ModelTransform transform) {
107         this.transform = transform;
108     }
109 }
```

55 com/sun/media/sound/ModelStandardDirector.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * A standard director who chooses performers
 * by there keyfrom,keyto,velfrom,velto properties.
 *
 * @author Karl Helgason
 */
public class ModelStandardDirector implements ModelDirector {

    ModelPerformer[] performers;
    ModelDirectedPlayer player;
    boolean noteOnUsed = false;
    boolean noteOffUsed = false;

    public ModelStandardDirector(ModelPerformer[] performers,
            ModelDirectedPlayer player) {
        this.performers = performers;
        this.player = player;
        for (int i = 0; i < performers.length; i++) {
            ModelPerformer p = performers[i];
            if (p.isReleaseTriggered()) {
                noteOffUsed = true;
            } else {
                noteOnUsed = true;
            }
        }
    }

    public void close() {
    }

    public void noteOff(int noteNumber, int velocity) {
        if (!noteOffUsed)
            return;
        for (int i = 0; i < performers.length; i++) {
```

```
            ModelPerformer p = performers[i];
            if (p.getKeyFrom() <= noteNumber && p.getKeyTo() >= noteNumber) {
                if (p.getVelFrom() <= velocity && p.getVelTo() >= velocity) {
                    if (p.isReleaseTriggered()) {
                        player.play(i, null);
                    }
                }
            }
        }
    }

    public void noteOn(int noteNumber, int velocity) {
        if (!noteOnUsed)
            return;
        for (int i = 0; i < performers.length; i++) {
            ModelPerformer p = performers[i];
            if (p.getKeyFrom() <= noteNumber && p.getKeyTo() >= noteNumber) {
                if (p.getVelFrom() <= velocity && p.getVelTo() >= velocity) {
                    if (!p.isReleaseTriggered()) {
                        player.play(i, null);
                    }
                }
            }
        }
    }
}
```

56 com/sun/media/sound/ModelStandardIndexedDirector.java

```java
/*
 * Copyright 2010 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * A standard indexed director who chooses performers
 * by there keyfrom,keyto,velfrom,velto properties.
 *
 * @author Karl Helgason
 */
public class ModelStandardIndexedDirector implements ModelDirector {

    ModelPerformer[] performers;
    ModelDirectedPlayer player;
    boolean noteOnUsed = false;
    boolean noteOffUsed = false;

    // Variables needed for index
    byte[][] trantables;
    int[] counters;
    int[][] mat;

    public ModelStandardIndexedDirector(ModelPerformer[] performers,
            ModelDirectedPlayer player) {
        this.performers = performers;
        this.player = player;
        for (int i = 0; i < performers.length; i++) {
            ModelPerformer p = performers[i];
            if (p.isReleaseTriggered()) {
                noteOffUsed = true;
            } else {
                noteOnUsed = true;
            }
        }
        buildindex();
    }

    private int[] lookupIndex(int x, int y) {
```

```
        if ((x >= 0) && (x < 128) && (y >= 0) && (y < 128)) {
            int xt = trantables[0][x];
            int yt = trantables[1][y];
            if (xt != -1 && yt != -1) {
                return mat[xt + yt * counters[0]];
            }
        }
        return null;
    }

    private int restrict(int value) {
        if(value < 0) return 0;
        if(value > 127) return 127;
        return value;
    }

    private void buildindex() {
        trantables = new byte[2][129];
        counters = new int[trantables.length];
        for (ModelPerformer performer : performers) {
            int keyFrom = performer.getKeyFrom();
            int keyTo = performer.getKeyTo();
            int velFrom = performer.getVelFrom();
            int velTo = performer.getVelTo();
            if (keyFrom > keyTo) continue;
            if (velFrom > velTo) continue;
            keyFrom = restrict(keyFrom);
            keyTo = restrict(keyTo);
            velFrom = restrict(velFrom);
            velTo = restrict(velTo);
            trantables[0][keyFrom] = 1;
            trantables[0][keyTo + 1] = 1;
            trantables[1][velFrom] = 1;
            trantables[1][velTo + 1] = 1;
        }
        for (int d = 0; d < trantables.length; d++) {
            byte[] trantable = trantables[d];
            int transize = trantable.length;
            for (int i = transize - 1; i >= 0; i--) {
                if (trantable[i] == 1) {
                    trantable[i] = -1;
                    break;
                }
                trantable[i] = -1;
            }
            int counter = -1;
            for (int i = 0; i < transize; i++) {
                if (trantable[i] != 0) {
                    counter++;
                    if (trantable[i] == -1)
                        break;
                }
                trantable[i] = (byte) counter;
            }
            counters[d] = counter;
        }
        mat = new int[counters[0] * counters[1]][];
        int ix = 0;
        for (ModelPerformer performer : performers) {
            int keyFrom = performer.getKeyFrom();
            int keyTo = performer.getKeyTo();
            int velFrom = performer.getVelFrom();
```

```
            int velTo = performer.getVelTo();
            if (keyFrom > keyTo) continue;
            if (velFrom > velTo) continue;
            keyFrom = restrict(keyFrom);
            keyTo = restrict(keyTo);
            velFrom = restrict(velFrom);
            velTo = restrict(velTo);
            int x_from = trantables[0][keyFrom];
            int x_to = trantables[0][keyTo + 1];
            int y_from = trantables[1][velFrom];
            int y_to = trantables[1][velTo + 1];
            if (x_to == -1)
                x_to = counters[0];
            if (y_to == -1)
                y_to = counters[1];
            for (int y = y_from; y < y_to; y++) {
                int i = x_from + y * counters[0];
                for (int x = x_from; x < x_to; x++) {
                    int[] mprev = mat[i];
                    if (mprev == null) {
                        mat[i] = new int[] { ix };
                    } else {
                        int[] mnew = new int[mprev.length + 1];
                        mnew[mnew.length - 1] = ix;
                        for (int k = 0; k < mprev.length; k++)
                            mnew[k] = mprev[k];
                        mat[i] = mnew;
                    }
                    i++;
                }
            }
            ix++;
        }
    }

    public void close() {
    }

    public void noteOff(int noteNumber, int velocity) {
        if (!noteOffUsed)
            return;
        int[] plist = lookupIndex(noteNumber, velocity);
        if(plist == null) return;
        for (int i : plist) {
            ModelPerformer p = performers[i];
            if (p.isReleaseTriggered()) {
                player.play(i, null);
            }
        }
    }

    public void noteOn(int noteNumber, int velocity) {
        if (!noteOnUsed)
            return;
        int[] plist = lookupIndex(noteNumber, velocity);
        if(plist == null) return;
        for (int i : plist) {
            ModelPerformer p = performers[i];
            if (!p.isReleaseTriggered()) {
                player.play(i, null);
            }
        }
```

```
185         }
186 }
```

57 com/sun/media/sound/ModelStandardTransform.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * A standard transformer used in connection blocks.
 * It expects input values to be between 0 and 1.
 *
 * The result of the transform is
 *   between 0 and 1 if polarity = unipolar and
 *   between -1 and 1 if polarity = bipolar.
 *
 * These constraints only applies to Concave, Convex and Switch transforms.
 *
 * @author Karl Helgason
 */
public class ModelStandardTransform implements ModelTransform {

    public static final boolean DIRECTION_MIN2MAX = false;
    public static final boolean DIRECTION_MAX2MIN = true;
    public static final boolean POLARITY_UNIPOLAR = false;
    public static final boolean POLARITY_BIPOLAR = true;
    public static final int TRANSFORM_LINEAR = 0;
    // concave: output = (20*log10(127^2/value^2)) / 96
    public static final int TRANSFORM_CONCAVE = 1;
    // convex: same as concave except that start and end point are reversed.
    public static final int TRANSFORM_CONVEX = 2;
    // switch: if value > avg(max,min) then max else min
    public static final int TRANSFORM_SWITCH = 3;
    public static final int TRANSFORM_ABSOLUTE = 4;
    private boolean direction = DIRECTION_MIN2MAX;
    private boolean polarity = POLARITY_UNIPOLAR;
    private int transform = TRANSFORM_LINEAR;

    public ModelStandardTransform() {
    }

    public ModelStandardTransform(boolean direction) {
```

```
        this.direction = direction;
    }

    public ModelStandardTransform(boolean direction, boolean polarity) {
        this.direction = direction;
        this.polarity = polarity;
    }

    public ModelStandardTransform(boolean direction, boolean polarity,
            int transform) {
        this.direction = direction;
        this.polarity = polarity;
        this.transform = transform;
    }

    public double transform(double value) {
        double s;
        double a;
        if (direction == DIRECTION_MAX2MIN)
            value = 1.0 - value;
        if (polarity == POLARITY_BIPOLAR)
            value = value * 2.0 - 1.0;
        switch (transform) {
            case TRANSFORM_CONCAVE:
                s = Math.signum(value);
                a = Math.abs(value);
                a = -((5.0 / 12.0) / Math.log(10)) * Math.log(1.0 - a);
                if (a < 0)
                    a = 0;
                else if (a > 1)
                    a = 1;
                return s * a;
            case TRANSFORM_CONVEX:
                s = Math.signum(value);
                a = Math.abs(value);
                a = 1.0 + ((5.0 / 12.0) / Math.log(10)) * Math.log(a);
                if (a < 0)
                    a = 0;
                else if (a > 1)
                    a = 1;
                return s * a;
            case TRANSFORM_SWITCH:
                if (polarity == POLARITY_BIPOLAR)
                    return (value > 0) ? 1 : -1;
                else
                    return (value > 0.5) ? 1 : 0;
            case TRANSFORM_ABSOLUTE:
                return Math.abs(value);
            default:
                break;
        }

        return value;
    }

    public boolean getDirection() {
        return direction;
    }

    public void setDirection(boolean direction) {
        this.direction = direction;
    }
```

```
123
124     public boolean getPolarity() {
125         return polarity;
126     }
127
128     public void setPolarity(boolean polarity) {
129         this.polarity = polarity;
130     }
131
132     public int getTransform() {
133         return transform;
134     }
135
136     public void setTransform(int transform) {
137         this.transform = transform;
138     }
139 }
```

58 com/sun/media/sound/ModelTransform.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * Model transform interface.
 *
 * @author Karl Helgason
 */
public interface ModelTransform {

    abstract public double transform(double value);
}
```

59 com/sun/media/sound/ModelWavetable.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * This is a wavetable oscillator interface.
 *
 * @author Karl Helgason
 */
public interface ModelWavetable extends ModelOscillator {

    public static final int LOOP_TYPE_OFF = 0;
    public static final int LOOP_TYPE_FORWARD = 1;
    public static final int LOOP_TYPE_RELEASE = 2;
    public static final int LOOP_TYPE_PINGPONG = 4;
    public static final int LOOP_TYPE_REVERSE = 8;

    public AudioFloatInputStream openStream();

    public float getLoopLength();

    public float getLoopStart();

    public int getLoopType();

    public float getPitchcorrection();
}
```

60 com/sun/media/sound/RIFFInvalidDataException.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * This exception is used when a RIFF file contains illegal or unexpected data.
 *
 * @author Karl Helgason
 */
public class RIFFInvalidDataException extends InvalidDataException {

    private static final long serialVersionUID = 1L;

    public RIFFInvalidDataException() {
        super("Invalid Data!");
    }

    public RIFFInvalidDataException(String s) {
        super(s);
    }
}
```

61 com/sun/media/sound/RIFFInvalidFormatException.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * This exception is used when a reader is used to read RIFF file of a format it
 * doesn't unterstand or support.
 *
 * @author Karl Helgason
 */
public class RIFFInvalidFormatException extends InvalidFormatException {

    private static final long serialVersionUID = 1L;

    public RIFFInvalidFormatException() {
        super("Invalid format!");
    }

    public RIFFInvalidFormatException(String s) {
        super(s);
    }
}
```

62 com/sun/media/sound/RIFFReader.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.io.EOFException;
import java.io.IOException;
import java.io.InputStream;

/**
 * Resource Interchange File Format (RIFF) stream decoder.
 *
 * @author Karl Helgason
 */
public class RIFFReader extends InputStream {

    private RIFFReader root;
    private long filepointer = 0;
    private String fourcc;
    private String riff_type = null;
    private long ckSize = 0;
    private InputStream stream;
    private long avail;
    private RIFFReader lastiterator = null;

    public RIFFReader(InputStream stream) throws IOException {

        if (stream instanceof RIFFReader)
            root = ((RIFFReader)stream).root;
        else
            root = this;

        this.stream = stream;
        avail = Integer.MAX_VALUE;
        ckSize = Integer.MAX_VALUE;

        // Check for RIFF null paddings,
        int b;
        while (true) {
```

```
            b = read();
            if (b == -1) {
                fourcc = ""; // don't put null value into fourcc,
                // because it is expected to
                // always contain a string value
                riff_type = null;
                avail = 0;
                return;
            }
            if (b != 0)
                break;
        }

        byte[] fourcc = new byte[4];
        fourcc[0] = (byte) b;
        readFully(fourcc, 1, 3);
        this.fourcc = new String(fourcc, "ascii");
        ckSize = readUnsignedInt();

        avail = this.ckSize;

        if (getFormat().equals("RIFF") || getFormat().equals("LIST")) {
            byte[] format = new byte[4];
            readFully(format);
            this.riff_type = new String(format, "ascii");
        }
    }

    public long getFilePointer() throws IOException {
        return root.filepointer;
    }

    public boolean hasNextChunk() throws IOException {
        if (lastiterator != null)
            lastiterator.finish();
        return avail != 0;
    }

    public RIFFReader nextChunk() throws IOException {
        if (lastiterator != null)
            lastiterator.finish();
        if (avail == 0)
            return null;
        lastiterator = new RIFFReader(this);
        return lastiterator;
    }

    public String getFormat() {
        return fourcc;
    }

    public String getType() {
        return riff_type;
    }

    public long getSize() {
        return ckSize;
    }

    public int read() throws IOException {
        if (avail == 0)
            return -1;
```

```
        int b = stream.read();
        if (b == -1)
            return -1;
        avail--;
        filepointer++;
        return b;
    }

    public int read(byte[] b, int offset, int len) throws IOException {
        if (avail == 0)
            return -1;
        if (len > avail) {
            int rlen = stream.read(b, offset, (int)avail);
            if (rlen != -1)
                filepointer += rlen;
            avail = 0;
            return rlen;
        } else {
            int ret = stream.read(b, offset, len);
            if (ret == -1)
                return -1;
            avail -= ret;
            filepointer += ret;
            return ret;
        }
    }

    public final void readFully(byte b[]) throws IOException {
        readFully(b, 0, b.length);
    }

    public final void readFully(byte b[], int off, int len) throws IOException {
        if (len < 0)
            throw new IndexOutOfBoundsException();
        while (len > 0) {
            int s = read(b, off, len);
            if (s < 0)
                throw new EOFException();
            if (s == 0)
                Thread.yield();
            off += s;
            len -= s;
        }
    }

    public final long skipBytes(long n) throws IOException {
        if (n < 0)
            return 0;
        long skipped = 0;
        while (skipped != n) {
            long s = skip(n - skipped);
            if (s < 0)
                break;
            if (s == 0)
                Thread.yield();
            skipped += s;
        }
        return skipped;
    }

    public long skip(long n) throws IOException {
        if (avail == 0)
```

```
            return -1;
        if (n > avail) {
            long len = stream.skip(avail);
            if (len != -1)
                filepointer += len;
            avail = 0;
            return len;
        } else {
            long ret = stream.skip(n);
            if (ret == -1)
                return -1;
            avail -= ret;
            filepointer += ret;
            return ret;
        }
    }

    public int available() {
        return (int)avail;
    }

    public void finish() throws IOException {
        if (avail != 0) {
            skipBytes(avail);
        }
    }

    // Read ASCII chars from stream
    public String readString(int len) throws IOException {
        byte[] buff = new byte[len];
        readFully(buff);
        for (int i = 0; i < buff.length; i++) {
            if (buff[i] == 0) {
                return new String(buff, 0, i, "ascii");
            }
        }
        return new String(buff, "ascii");
    }

    // Read 8 bit signed integer from stream
    public byte readByte() throws IOException {
        int ch = read();
        if (ch < 0)
            throw new EOFException();
        return (byte) ch;
    }

    // Read 16 bit signed integer from stream
    public short readShort() throws IOException {
        int ch1 = read();
        int ch2 = read();
        if (ch1 < 0)
            throw new EOFException();
        if (ch2 < 0)
            throw new EOFException();
        return (short)(ch1 | (ch2 << 8));
    }

    // Read 32 bit signed integer from stream
    public int readInt() throws IOException {
        int ch1 = read();
        int ch2 = read();
```

```
        int ch3 = read();
        int ch4 = read();
        if (ch1 < 0)
            throw new EOFException();
        if (ch2 < 0)
            throw new EOFException();
        if (ch3 < 0)
            throw new EOFException();
        if (ch4 < 0)
            throw new EOFException();
        return ch1 + (ch2 << 8) | (ch3 << 16) | (ch4 << 24);
    }

    // Read 64 bit signed integer from stream
    public long readLong() throws IOException {
        long ch1 = read();
        long ch2 = read();
        long ch3 = read();
        long ch4 = read();
        long ch5 = read();
        long ch6 = read();
        long ch7 = read();
        long ch8 = read();
        if (ch1 < 0)
            throw new EOFException();
        if (ch2 < 0)
            throw new EOFException();
        if (ch3 < 0)
            throw new EOFException();
        if (ch4 < 0)
            throw new EOFException();
        if (ch5 < 0)
            throw new EOFException();
        if (ch6 < 0)
            throw new EOFException();
        if (ch7 < 0)
            throw new EOFException();
        if (ch8 < 0)
            throw new EOFException();
        return ch1 | (ch2 << 8) | (ch3 << 16) | (ch4 << 24)
                | (ch5 << 32) | (ch6 << 40) | (ch7 << 48) | (ch8 << 56);
    }

    // Read 8 bit unsigned integer from stream
    public int readUnsignedByte() throws IOException {
        int ch = read();
        if (ch < 0)
            throw new EOFException();
        return ch;
    }

    // Read 16 bit unsigned integer from stream
    public int readUnsignedShort() throws IOException {
        int ch1 = read();
        int ch2 = read();
        if (ch1 < 0)
            throw new EOFException();
        if (ch2 < 0)
            throw new EOFException();
        return ch1 | (ch2 << 8);
    }

```

```
    // Read 32 bit unsigned integer from stream
    public long readUnsignedInt() throws IOException {
        long ch1 = read();
        long ch2 = read();
        long ch3 = read();
        long ch4 = read();
        if (ch1 < 0)
            throw new EOFException();
        if (ch2 < 0)
            throw new EOFException();
        if (ch3 < 0)
            throw new EOFException();
        if (ch4 < 0)
            throw new EOFException();
        return ch1 + (ch2 << 8) | (ch3 << 16) | (ch4 << 24);
    }

    public void close() throws IOException {
        finish();
        if (this == root)
            stream.close();
        stream = null;
    }
}
```

63 com/sun/media/sound/RIFFWriter.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.OutputStream;
import java.io.RandomAccessFile;

/**
 * Resource Interchange File Format (RIFF) stream encoder.
 *
 * @author Karl Helgason
 */
public class RIFFWriter extends OutputStream {

    private interface RandomAccessWriter {

        public void seek(long chunksizepointer) throws IOException;

        public long getPointer() throws IOException;

        public void close() throws IOException;

        public void write(int b) throws IOException;

        public void write(byte[] b, int off, int len) throws IOException;

        public void write(byte[] bytes) throws IOException;

        public long length() throws IOException;

        public void setLength(long i) throws IOException;
    }

    private static class RandomAccessFileWriter implements RandomAccessWriter {

```

```
61          RandomAccessFile raf;
62
63          public RandomAccessFileWriter(File file) throws FileNotFoundException {
64              this.raf = new RandomAccessFile(file, "rw");
65          }
66
67          public RandomAccessFileWriter(String name) throws FileNotFoundException {
68              this.raf = new RandomAccessFile(name, "rw");
69          }
70
71          public void seek(long chunksizepointer) throws IOException {
72              raf.seek(chunksizepointer);
73          }
74
75          public long getPointer() throws IOException {
76              return raf.getFilePointer();
77          }
78
79          public void close() throws IOException {
80              raf.close();
81          }
82
83          public void write(int b) throws IOException {
84              raf.write(b);
85          }
86
87          public void write(byte[] b, int off, int len) throws IOException {
88              raf.write(b, off, len);
89          }
90
91          public void write(byte[] bytes) throws IOException {
92              raf.write(bytes);
93          }
94
95          public long length() throws IOException {
96              return raf.length();
97          }
98
99          public void setLength(long i) throws IOException {
100             raf.setLength(i);
101         }
102     }
103
104     private static class RandomAccessByteWriter implements RandomAccessWriter {
105
106         byte[] buff = new byte[32];
107         int length = 0;
108         int pos = 0;
109         byte[] s;
110         OutputStream stream;
111
112         public RandomAccessByteWriter(OutputStream stream) {
113             this.stream = stream;
114         }
115
116         public void seek(long chunksizepointer) throws IOException {
117             pos = (int) chunksizepointer;
118         }
119
120         public long getPointer() throws IOException {
121             return pos;
122         }
```

```
123
124         public void close() throws IOException {
125             stream.write(buff, 0, length);
126             stream.close();
127         }
128
129         public void write(int b) throws IOException {
130             if (s == null)
131                 s = new byte[1];
132             s[0] = (byte)b;
133             write(s, 0, 1);
134         }
135
136         public void write(byte[] b, int off, int len) throws IOException {
137             int newsize = pos + len;
138             if (newsize > length)
139                 setLength(newsize);
140             int end = off + len;
141             for (int i = off; i < end; i++) {
142                 buff[pos++] = b[i];
143             }
144         }
145
146         public void write(byte[] bytes) throws IOException {
147             write(bytes, 0, bytes.length);
148         }
149
150         public long length() throws IOException {
151             return length;
152         }
153
154         public void setLength(long i) throws IOException {
155             length = (int) i;
156             if (length > buff.length) {
157                 int newlen = Math.max(buff.length << 1, length);
158                 byte[] newbuff = new byte[newlen];
159                 System.arraycopy(buff, 0, newbuff, 0, buff.length);
160                 buff = newbuff;
161             }
162         }
163     }
164     private int chunktype = 0; // 0=RIFF, 1=LIST; 2=CHUNK
165     private RandomAccessWriter raf;
166     private long chunksizepointer;
167     private long startpointer;
168     private RIFFWriter childchunk = null;
169     private boolean open = true;
170     private boolean writeoverride = false;
171
172     public RIFFWriter(String name, String format) throws IOException {
173         this(new RandomAccessFileWriter(name), format, 0);
174     }
175
176     public RIFFWriter(File file, String format) throws IOException {
177         this(new RandomAccessFileWriter(file), format, 0);
178     }
179
180     public RIFFWriter(OutputStream stream, String format) throws IOException {
181         this(new RandomAccessByteWriter(stream), format, 0);
182     }
183
184     private RIFFWriter(RandomAccessWriter raf, String format, int chunktype)
```

```
            throws IOException {
        if (chunktype == 0)
            if (raf.length() != 0)
                raf.setLength(0);
        this.raf = raf;
        if (raf.getPointer() % 2 != 0)
            raf.write(0);

        if (chunktype == 0)
            raf.write("RIFF".getBytes("ascii"));
        else if (chunktype == 1)
            raf.write("LIST".getBytes("ascii"));
        else
            raf.write((format + "    ").substring(0, 4).getBytes("ascii"));

        chunksizepointer = raf.getPointer();
        this.chunktype = 2;
        writeUnsignedInt(0);
        this.chunktype = chunktype;
        startpointer = raf.getPointer();
        if (chunktype != 2)
            raf.write((format + "    ").substring(0, 4).getBytes("ascii"));

    }

    public void seek(long pos) throws IOException {
        raf.seek(pos);
    }

    public long getFilePointer() throws IOException {
        return raf.getPointer();
    }

    public void setWriteOverride(boolean writeoverride) {
        this.writeoverride = writeoverride;
    }

    public boolean getWriteOverride() {
        return writeoverride;
    }

    public void close() throws IOException {
        if (!open)
            return;
        if (childchunk != null) {
            childchunk.close();
            childchunk = null;
        }

        int bakchunktype = chunktype;
        long fpointer = raf.getPointer();
        raf.seek(chunksizepointer);
        chunktype = 2;
        writeUnsignedInt(fpointer - startpointer);

        if (bakchunktype == 0)
            raf.close();
        else
            raf.seek(fpointer);
        open = false;
        raf = null;
    }
```

```
247
248     public void write(int b) throws IOException {
249         if (!writeoverride) {
250             if (chunktype != 2) {
251                 throw new IllegalArgumentException(
252                         "Only chunks can write bytes!");
253             }
254             if (childchunk != null) {
255                 childchunk.close();
256                 childchunk = null;
257             }
258         }
259         raf.write(b);
260     }
261
262     public void write(byte b[], int off, int len) throws IOException {
263         if (!writeoverride) {
264             if (chunktype != 2) {
265                 throw new IllegalArgumentException(
266                         "Only chunks can write bytes!");
267             }
268             if (childchunk != null) {
269                 childchunk.close();
270                 childchunk = null;
271             }
272         }
273         raf.write(b, off, len);
274     }
275
276     public RIFFWriter writeList(String format) throws IOException {
277         if (chunktype == 2) {
278             throw new IllegalArgumentException(
279                     "Only LIST and RIFF can write lists!");
280         }
281         if (childchunk != null) {
282             childchunk.close();
283             childchunk = null;
284         }
285         childchunk = new RIFFWriter(this.raf, format, 1);
286         return childchunk;
287     }
288
289     public RIFFWriter writeChunk(String format) throws IOException {
290         if (chunktype == 2) {
291             throw new IllegalArgumentException(
292                     "Only LIST and RIFF can write chunks!");
293         }
294         if (childchunk != null) {
295             childchunk.close();
296             childchunk = null;
297         }
298         childchunk = new RIFFWriter(this.raf, format, 2);
299         return childchunk;
300     }
301
302     // Write ASCII chars to stream
303     public void writeString(String string) throws IOException {
304         byte[] buff = string.getBytes();
305         write(buff);
306     }
307
308     // Write ASCII chars to stream
```

```
    public void writeString(String string, int len) throws IOException {
        byte[] buff = string.getBytes();
        if (buff.length > len)
            write(buff, 0, len);
        else {
            write(buff);
            for (int i = buff.length; i < len; i++)
                write(0);
        }
    }

    // Write 8 bit signed integer to stream
    public void writeByte(int b) throws IOException {
        write(b);
    }

    // Write 16 bit signed integer to stream
    public void writeShort(short b) throws IOException {
        write((b >>> 0) & 0xFF);
        write((b >>> 8) & 0xFF);
    }

    // Write 32 bit signed integer to stream
    public void writeInt(int b) throws IOException {
        write((b >>> 0) & 0xFF);
        write((b >>> 8) & 0xFF);
        write((b >>> 16) & 0xFF);
        write((b >>> 24) & 0xFF);
    }

    // Write 64 bit signed integer to stream
    public void writeLong(long b) throws IOException {
        write((int) (b >>> 0) & 0xFF);
        write((int) (b >>> 8) & 0xFF);
        write((int) (b >>> 16) & 0xFF);
        write((int) (b >>> 24) & 0xFF);
        write((int) (b >>> 32) & 0xFF);
        write((int) (b >>> 40) & 0xFF);
        write((int) (b >>> 48) & 0xFF);
        write((int) (b >>> 56) & 0xFF);
    }

    // Write 8 bit unsigned integer to stream
    public void writeUnsignedByte(int b) throws IOException {
        writeByte((byte) b);
    }

    // Write 16 bit unsigned integer to stream
    public void writeUnsignedShort(int b) throws IOException {
        writeShort((short) b);
    }

    // Write 32 bit unsigned integer to stream
    public void writeUnsignedInt(long b) throws IOException {
        writeInt((int) b);
    }
}
```

64 com/sun/media/sound/RealTimeSequencer.java

```
/*
 * Copyright (c) 2003, 2011, Oracle and/or its affiliates. All rights reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Oracle designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Oracle in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
 * or visit www.oracle.com if you need additional information or have any
 * questions.
 */

package com.sun.media.sound;

import java.io.ByteArrayOutputStream;
import java.io.ByteArrayInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStream;

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

import javax.sound.midi.*;


/**
 * A Real Time Sequencer
 *
 * @author Florian Bomers
 */

/* TODO:
 * - rename PlayThread to PlayEngine (because isn't a thread)
 */
class RealTimeSequencer extends AbstractMidiDevice implements Sequencer, AutoConnectSequencer {

    // STATIC VARIABLES

    /** debugging flags */
    private final static boolean DEBUG_PUMP = false;
    private final static boolean DEBUG_PUMP_ALL = false;

    /**
     * Event Dispatcher thread. Should be using a shared event
     * dispatcher instance with a factory in EventDispatcher
     */
```

```
61      private static final EventDispatcher eventDispatcher;
62
63      /**
64       * All RealTimeSequencers share this info object.
65       */
66      static final RealTimeSequencerInfo info = new RealTimeSequencerInfo();
67
68
69      private static Sequencer.SyncMode[] masterSyncModes = { Sequencer.SyncMode.INTERNAL_CLOCK };
70      private static Sequencer.SyncMode[] slaveSyncModes  = { Sequencer.SyncMode.NO_SYNC };
71
72      private static Sequencer.SyncMode masterSyncMode    = Sequencer.SyncMode.INTERNAL_CLOCK;
73      private static Sequencer.SyncMode slaveSyncMode     = Sequencer.SyncMode.NO_SYNC;
74
75
76      /**
77       * Sequence on which this sequencer is operating.
78       */
79      private Sequence sequence = null;
80
81      // caches
82
83      /**
84       * Same for setTempoInMPQ...
85       * -1 means not set.
86       */
87      private double cacheTempoMPQ = -1;
88
89
90      /**
91       * cache value for tempo factor until sequence is set
92       * -1 means not set.
93       */
94      private float cacheTempoFactor = -1;
95
96
97      /** if a particular track is muted */
98      private boolean[] trackMuted = null;
99      /** if a particular track is solo */
100     private boolean[] trackSolo = null;
101
102     /** tempo cache for getMicrosecondPosition */
103     private MidiUtils.TempoCache tempoCache = new MidiUtils.TempoCache();
104
105     /**
106      * True if the sequence is running.
107      */
108     private boolean running = false;
109
110
111     /** the thread for pushing out the MIDI messages */
112     private PlayThread playThread;
113
114
115     /**
116      * True if we are recording
117      */
118     private boolean recording = false;
119
120
121     /**
122      * List of tracks to which we're recording
```

```
123         */
124        private List recordingTracks = new ArrayList();
125
126
127        private long loopStart = 0;
128        private long loopEnd = -1;
129        private int loopCount = 0;
130
131
132        /**
133         * Meta event listeners
134         */
135        private ArrayList metaEventListeners = new ArrayList();
136
137
138        /**
139         * Control change listeners
140         */
141        private ArrayList controllerEventListeners = new ArrayList();
142
143
144        /** automatic connection support */
145        private boolean autoConnect = false;
146
147        /** if we need to autoconnect at next open */
148        private boolean doAutoConnectAtNextOpen = false;
149
150        /** the receiver that this device is auto-connected to */
151        Receiver autoConnectedReceiver = null;
152
153
154        static {
155            // create and start the global event thread
156            eventDispatcher = new EventDispatcher();
157            eventDispatcher.start();
158        }
159
160
161        /* ***************************** CONSTRUCTOR ***************************** */
162
163        protected RealTimeSequencer() throws MidiUnavailableException {
164            super(info);
165
166            if (Printer.trace) Printer.trace(">> RealTimeSequencer CONSTRUCTOR");
167            if (Printer.trace) Printer.trace("<< RealTimeSequencer CONSTRUCTOR completed");
168        }
169
170
171        /* ***************************** SEQUENCER METHODS ******************** */
172
173        public synchronized void setSequence(Sequence sequence)
174            throws InvalidMidiDataException {
175
176            if (Printer.trace) Printer.trace(">> RealTimeSequencer: setSequence(" + sequence +")");
177
178            if (sequence != this.sequence) {
179                if (this.sequence != null && sequence == null) {
180                    setCaches();
181                    stop();
182                    // initialize some non-cached values
183                    trackMuted = null;
184                    trackSolo = null;
```

```
185                 loopStart = 0;
186                 loopEnd = -1;
187                 loopCount = 0;
188                 if (getDataPump() != null) {
189                     getDataPump().setTickPos(0);
190                     getDataPump().resetLoopCount();
191                 }
192             }
193 
194             if (playThread != null) {
195                 playThread.setSequence(sequence);
196             }
197 
198             // store this sequence (do not copy - we want to give the possibility
199             // of modifying the sequence at runtime)
200             this.sequence = sequence;
201 
202             if (sequence != null) {
203                 tempoCache.refresh(sequence);
204                 // rewind to the beginning
205                 setTickPosition(0);
206                 // propagate caches
207                 propagateCaches();
208             }
209         }
210         else if (sequence != null) {
211             tempoCache.refresh(sequence);
212             if (playThread != null) {
213                 playThread.setSequence(sequence);
214             }
215         }
216 
217         if (Printer.trace) Printer.trace("<< RealTimeSequencer: setSequence(" + sequence +") 
                completed");
218     }
219 
220 
221     public synchronized void setSequence(InputStream stream) throws IOException, 
            InvalidMidiDataException {
222 
223         if (Printer.trace) Printer.trace(">> RealTimeSequencer: setSequence(" + stream +")");
224 
225         if (stream == null) {
226             setSequence((Sequence) null);
227             return;
228         }
229 
230         Sequence seq = MidiSystem.getSequence(stream); // can throw IOException, 
                InvalidMidiDataException
231 
232         setSequence(seq);
233 
234         if (Printer.trace) Printer.trace("<< RealTimeSequencer: setSequence(" + stream +") 
                completed");
235 
236     }
237 
238 
239     public Sequence getSequence() {
240         return sequence;
241     }
242 
```

```
243
244     public synchronized void start() {
245         if (Printer.trace) Printer.trace(">> RealTimeSequencer: start()");
246
247         // sequencer not open: throw an exception
248         if (!isOpen()) {
249             throw new IllegalStateException("sequencer not open");
250         }
251
252         // sequence not available: throw an exception
253         if (sequence == null) {
254             throw new IllegalStateException("sequence not set");
255         }
256
257         // already running: return quietly
258         if (running == true) {
259             return;
260         }
261
262         // start playback
263         implStart();
264
265         if (Printer.trace) Printer.trace("<< RealTimeSequencer: start() completed");
266     }
267
268
269     public synchronized void stop() {
270         if (Printer.trace) Printer.trace(">> RealTimeSequencer: stop()");
271
272         if (!isOpen()) {
273             throw new IllegalStateException("sequencer not open");
274         }
275         stopRecording();
276
277         // not running; just return
278         if (running == false) {
279             if (Printer.trace) Printer.trace("<< RealTimeSequencer: stop() not running!");
280             return;
281         }
282
283         // stop playback
284         implStop();
285
286         if (Printer.trace) Printer.trace("<< RealTimeSequencer: stop() completed");
287     }
288
289
290     public boolean isRunning() {
291         return running;
292     }
293
294
295     public void startRecording() {
296         if (!isOpen()) {
297             throw new IllegalStateException("Sequencer not open");
298         }
299
300         start();
301         recording = true;
302     }
303
304
```

```
305     public void stopRecording() {
306         if (!isOpen()) {
307             throw new IllegalStateException("Sequencer not open");
308         }
309         recording = false;
310     }
311 
312 
313     public boolean isRecording() {
314         return recording;
315     }
316 
317 
318     public void recordEnable(Track track, int channel) {
319         if (!findTrack(track)) {
320             throw new IllegalArgumentException("Track does not exist in the current sequence");
321         }
322 
323         synchronized(recordingTracks) {
324             RecordingTrack rc = RecordingTrack.get(recordingTracks, track);
325             if (rc != null) {
326                 rc.channel = channel;
327             } else {
328                 recordingTracks.add(new RecordingTrack(track, channel));
329             }
330         }
331 
332     }
333 
334 
335     public void recordDisable(Track track) {
336         synchronized(recordingTracks) {
337             RecordingTrack rc = RecordingTrack.get(recordingTracks, track);
338             if (rc != null) {
339                 recordingTracks.remove(rc);
340             }
341         }
342 
343     }
344 
345 
346     private boolean findTrack(Track track) {
347         boolean found = false;
348         if (sequence != null) {
349             Track[] tracks = sequence.getTracks();
350             for (int i = 0; i < tracks.length; i++) {
351                 if (track == tracks[i]) {
352                     found = true;
353                     break;
354                 }
355             }
356         }
357         return found;
358     }
359 
360 
361     public float getTempoInBPM() {
362         if (Printer.trace) Printer.trace(">> RealTimeSequencer: getTempoInBPM() ");
363 
364         return (float) MidiUtils.convertTempo(getTempoInMPQ());
365     }
366 
```

```
367
368     public void setTempoInBPM(float bpm) {
369         if (Printer.trace) Printer.trace(">> RealTimeSequencer: setTempoInBPM() ");
370         if (bpm <= 0) {
371             // should throw IllegalArgumentException
372             bpm = 1.0f;
373         }
374
375         setTempoInMPQ((float) MidiUtils.convertTempo((double) bpm));
376     }
377
378
379     public float getTempoInMPQ() {
380         if (Printer.trace) Printer.trace(">> RealTimeSequencer: getTempoInMPQ() ");
381
382         if (needCaching()) {
383             // if the sequencer is closed, return cached value
384             if (cacheTempoMPQ != -1) {
385                 return (float) cacheTempoMPQ;
386             }
387             // if sequence is set, return current tempo
388             if (sequence != null) {
389                 return tempoCache.getTempoMPQAt(getTickPosition());
390             }
391
392             // last resort: return a standard tempo: 120bpm
393             return (float) MidiUtils.DEFAULT_TEMPO_MPQ;
394         }
395         return (float)getDataPump().getTempoMPQ();
396     }
397
398
399     public void setTempoInMPQ(float mpq) {
400         if (mpq <= 0) {
401             // should throw IllegalArgumentException
402             mpq = 1.0f;
403         }
404
405         if (Printer.trace) Printer.trace(">> RealTimeSequencer: setTempoInMPQ() ");
406
407         if (needCaching()) {
408             // cache the value
409             cacheTempoMPQ = mpq;
410         } else {
411             // set the native tempo in MPQ
412             getDataPump().setTempoMPQ(mpq);
413
414             // reset the tempoInBPM and tempoInMPQ values so we won't use them again
415             cacheTempoMPQ = -1;
416         }
417     }
418
419
420     public void setTempoFactor(float factor) {
421         if (factor <= 0) {
422             // should throw IllegalArgumentException
423             return;
424         }
425
426         if (Printer.trace) Printer.trace(">> RealTimeSequencer: setTempoFactor() ");
427
428         if (needCaching()) {
```

```
            cacheTempoFactor = factor;
        } else {
            getDataPump().setTempoFactor(factor);
            // don't need cache anymore
            cacheTempoFactor = -1;
        }
    }


    public float getTempoFactor() {
        if (Printer.trace) Printer.trace(">> RealTimeSequencer: getTempoFactor() ");

        if (needCaching()) {
            if (cacheTempoFactor != -1) {
                return cacheTempoFactor;
            }
            return 1.0f;
        }
        return getDataPump().getTempoFactor();
    }


    public long getTickLength() {
        if (Printer.trace) Printer.trace(">> RealTimeSequencer: getTickLength() ");

        if (sequence == null) {
            return 0;
        }

        return sequence.getTickLength();
    }


    public synchronized long getTickPosition() {
        if (Printer.trace) Printer.trace(">> RealTimeSequencer: getTickPosition() ");

        if (getDataPump() == null || sequence == null) {
            return 0;
        }

        return getDataPump().getTickPos();
    }


    public synchronized void setTickPosition(long tick) {
        if (tick < 0) {
            // should throw IllegalArgumentException
            return;
        }

        if (Printer.trace) Printer.trace(">> RealTimeSequencer: setTickPosition("+tick+") ");

        if (getDataPump() == null) {
            if (tick != 0) {
                // throw new InvalidStateException("cannot set position in closed state");
            }
        }
        else if (sequence == null) {
            if (tick != 0) {
                // throw new InvalidStateException("cannot set position if sequence is not set");
            }
        } else {
```

```
            getDataPump().setTickPos(tick);
        }
    }


    public long getMicrosecondLength() {
        if (Printer.trace) Printer.trace(">> RealTimeSequencer: getMicrosecondLength() ");

        if (sequence == null) {
            return 0;
        }

        return sequence.getMicrosecondLength();
    }


    public long getMicrosecondPosition() {
        if (Printer.trace) Printer.trace(">> RealTimeSequencer: getMicrosecondPosition() ");

        if (getDataPump() == null || sequence == null) {
            return 0;
        }
        synchronized (tempoCache) {
            return MidiUtils.tick2microsecond(sequence, getDataPump().getTickPos(), tempoCache);
        }
    }


    public void setMicrosecondPosition(long microseconds) {
        if (microseconds < 0) {
            // should throw IllegalArgumentException
            return;
        }

        if (Printer.trace) Printer.trace(">> RealTimeSequencer: setMicrosecondPosition("+
            microseconds+") ");

        if (getDataPump() == null) {
            if (microseconds != 0) {
                // throw new InvalidStateException("cannot set position in closed state");
            }
        }
        else if (sequence == null) {
            if (microseconds != 0) {
                // throw new InvalidStateException("cannot set position if sequence is not set");
            }
        } else {
            synchronized(tempoCache) {
                setTickPosition(MidiUtils.microsecond2tick(sequence, microseconds, tempoCache));
            }
        }
    }


    public void setMasterSyncMode(Sequencer.SyncMode sync) {
        // not supported
    }


    public Sequencer.SyncMode getMasterSyncMode() {
        return masterSyncMode;
    }
```

```
552
553
554     public Sequencer.SyncMode[] getMasterSyncModes() {
555         Sequencer.SyncMode[] returnedModes = new Sequencer.SyncMode[masterSyncModes.length];
556         System.arraycopy(masterSyncModes, 0, returnedModes, 0, masterSyncModes.length);
557         return returnedModes;
558     }
559
560
561     public void setSlaveSyncMode(Sequencer.SyncMode sync) {
562         // not supported
563     }
564
565
566     public Sequencer.SyncMode getSlaveSyncMode() {
567         return slaveSyncMode;
568     }
569
570
571     public Sequencer.SyncMode[] getSlaveSyncModes() {
572         Sequencer.SyncMode[] returnedModes = new Sequencer.SyncMode[slaveSyncModes.length];
573         System.arraycopy(slaveSyncModes, 0, returnedModes, 0, slaveSyncModes.length);
574         return returnedModes;
575     }
576
577     protected int getTrackCount() {
578         Sequence seq = getSequence();
579         if (seq != null) {
580             // $$fb wish there was a nicer way to get the number of tracks...
581             return sequence.getTracks().length;
582         }
583         return 0;
584     }
585
586
587
588     public synchronized void setTrackMute(int track, boolean mute) {
589         int trackCount = getTrackCount();
590         if (track < 0 || track >= getTrackCount()) return;
591         trackMuted = ensureBoolArraySize(trackMuted, trackCount);
592         trackMuted[track] = mute;
593         if (getDataPump() != null) {
594             getDataPump().muteSoloChanged();
595         }
596     }
597
598
599     public synchronized boolean getTrackMute(int track) {
600         if (track < 0 || track >= getTrackCount()) return false;
601         if (trackMuted == null || trackMuted.length <= track) return false;
602         return trackMuted[track];
603     }
604
605
606     public synchronized void setTrackSolo(int track, boolean solo) {
607         int trackCount = getTrackCount();
608         if (track < 0 || track >= getTrackCount()) return;
609         trackSolo = ensureBoolArraySize(trackSolo, trackCount);
610         trackSolo[track] = solo;
611         if (getDataPump() != null) {
612             getDataPump().muteSoloChanged();
613         }
```

```
614     }
615
616
617     public synchronized boolean getTrackSolo(int track) {
618         if (track < 0 || track >= getTrackCount()) return false;
619         if (trackSolo == null || trackSolo.length <= track) return false;
620         return trackSolo[track];
621     }
622
623
624     public boolean addMetaEventListener(MetaEventListener listener) {
625         synchronized(metaEventListeners) {
626             if (! metaEventListeners.contains(listener)) {
627
628                 metaEventListeners.add(listener);
629             }
630             return true;
631         }
632     }
633
634
635     public void removeMetaEventListener(MetaEventListener listener) {
636         synchronized(metaEventListeners) {
637             int index = metaEventListeners.indexOf(listener);
638             if (index >= 0) {
639                 metaEventListeners.remove(index);
640             }
641         }
642     }
643
644
645     public int[] addControllerEventListener(ControllerEventListener listener, int[] controllers)
        {
646         synchronized(controllerEventListeners) {
647
648             // first find the listener.  if we have one, add the controllers
649             // if not, create a new element for it.
650             ControllerListElement cve = null;
651             boolean flag = false;
652             for(int i=0; i < controllerEventListeners.size(); i++) {
653
654                 cve = (ControllerListElement) controllerEventListeners.get(i);
655
656                 if (cve.listener.equals(listener)) {
657                     cve.addControllers(controllers);
658                     flag = true;
659                     break;
660                 }
661             }
662             if (!flag) {
663                 cve = new ControllerListElement(listener, controllers);
664                 controllerEventListeners.add(cve);
665             }
666
667             // and return all the controllers this listener is interested in
668             return cve.getControllers();
669         }
670     }
671
672
673     public int[] removeControllerEventListener(ControllerEventListener listener, int[]
        controllers) {
```

```
        synchronized(controllerEventListeners) {
            ControllerListElement cve = null;
            boolean flag = false;
            for (int i=0; i < controllerEventListeners.size(); i++) {
                cve = (ControllerListElement) controllerEventListeners.get(i);
                if (cve.listener.equals(listener)) {
                    cve.removeControllers(controllers);
                    flag = true;
                    break;
                }
            }
            if (!flag) {
                return new int[0];
            }
            if (controllers == null) {
                int index = controllerEventListeners.indexOf(cve);
                if (index >= 0) {
                    controllerEventListeners.remove(index);
                }
                return new int[0];
            }
            return cve.getControllers();
        }
    }


    ////////////////// LOOPING (added in 1.5) ////////////////////

    public void setLoopStartPoint(long tick) {
        if ((tick > getTickLength())
            || ((loopEnd != -1) && (tick > loopEnd))
            || (tick < 0)) {
            throw new IllegalArgumentException("invalid loop start point: "+tick);
        }
        loopStart = tick;
    }

    public long getLoopStartPoint() {
        return loopStart;
    }

    public void setLoopEndPoint(long tick) {
        if ((tick > getTickLength())
            || ((loopStart > tick) && (tick != -1))
            || (tick < -1)) {
            throw new IllegalArgumentException("invalid loop end point: "+tick);
        }
        loopEnd = tick;
    }

    public long getLoopEndPoint() {
        return loopEnd;
    }

    public void setLoopCount(int count) {
        if (count != LOOP_CONTINUOUSLY
            && count < 0) {
            throw new IllegalArgumentException("illegal value for loop count: "+count);
        }
        loopCount = count;
        if (getDataPump() != null) {
            getDataPump().resetLoopCount();
```

```
736         }
737     }
738 
739     public int getLoopCount() {
740         return loopCount;
741     }
742 
743 
744     /* ********************************** play control ************************* */
745 
746     /*
747      */
748     protected void implOpen() throws MidiUnavailableException {
749         if (Printer.trace) Printer.trace(">> RealTimeSequencer: implOpen()");
750 
751         //openInternalSynth();
752 
753         // create PlayThread
754         playThread = new PlayThread();
755 
756         //id = nOpen();
757         //if (id == 0) {
758         //    throw new MidiUnavailableException("unable to open sequencer");
759         //}
760         if (sequence != null) {
761             playThread.setSequence(sequence);
762         }
763 
764         // propagate caches
765         propagateCaches();
766 
767         if (doAutoConnectAtNextOpen) {
768             doAutoConnect();
769         }
770         if (Printer.trace) Printer.trace("<< RealTimeSequencer: implOpen() succeeded");
771     }
772 
773     private void doAutoConnect() {
774         if (Printer.trace) Printer.trace(">> RealTimeSequencer: doAutoConnect()");
775         Receiver rec = null;
776         // first try to connect to the default synthesizer
777         // IMPORTANT: this code needs to be synch'ed with
778         //            MidiSystem.getSequencer(boolean), because the same
779         //            algorithm needs to be used!
780         try {
781             Synthesizer synth = MidiSystem.getSynthesizer();
782             if (synth instanceof ReferenceCountingDevice) {
783                 rec = ((ReferenceCountingDevice) synth).getReceiverReferenceCounting();
784             } else {
785                 synth.open();
786                 try {
787                     rec = synth.getReceiver();
788                 } finally {
789                     // make sure that the synth is properly closed
790                     if (rec == null) {
791                         synth.close();
792                     }
793                 }
794             }
795         } catch (Exception e) {
796             // something went wrong with synth
797         }
```

```
798         if (rec == null) {
799             // then try to connect to the default Receiver
800             try {
801                 rec = MidiSystem.getReceiver();
802             } catch (Exception e) {
803                 // something went wrong. Nothing to do then!
804             }
805         }
806         if (rec != null) {
807             autoConnectedReceiver = rec;
808             try {
809                 getTransmitter().setReceiver(rec);
810             } catch (Exception e) {}
811         }
812         if (Printer.trace) Printer.trace("<< RealTimeSequencer: doAutoConnect() succeeded");
813     }
814 
815     private synchronized void propagateCaches() {
816         // only set caches if open and sequence is set
817         if (sequence != null && isOpen()) {
818             if (cacheTempoFactor != -1) {
819                 setTempoFactor(cacheTempoFactor);
820             }
821             if (cacheTempoMPQ == -1) {
822                 setTempoInMPQ((new MidiUtils.TempoCache(sequence)).getTempoMPQAt(getTickPosition
                        ()));
823             } else {
824                 setTempoInMPQ((float) cacheTempoMPQ);
825             }
826         }
827     }
828 
829     /** populate the caches with the current values */
830     private synchronized void setCaches() {
831         cacheTempoFactor = getTempoFactor();
832         cacheTempoMPQ = getTempoInMPQ();
833     }
834 
835 
836 
837     protected synchronized void implClose() {
838         if (Printer.trace) Printer.trace(">> RealTimeSequencer: implClose() ");
839 
840         if (playThread == null) {
841             if (Printer.err) Printer.err("RealTimeSequencer.implClose() called, but playThread
                    not instanciated!");
842         } else {
843             // Interrupt playback loop.
844             playThread.close();
845             playThread = null;
846         }
847 
848         super.implClose();
849 
850         sequence = null;
851         running = false;
852         cacheTempoMPQ = -1;
853         cacheTempoFactor = -1;
854         trackMuted = null;
855         trackSolo = null;
856         loopStart = 0;
857         loopEnd = -1;
```

```
858         loopCount = 0;
859 
860         /** if this sequencer is set to autoconnect, need to
861          * re-establish the connection at next open!
862          */
863         doAutoConnectAtNextOpen = autoConnect;
864 
865         if (autoConnectedReceiver != null) {
866             try {
867                 autoConnectedReceiver.close();
868             } catch (Exception e) {}
869             autoConnectedReceiver = null;
870         }
871 
872         if (Printer.trace) Printer.trace("<< RealTimeSequencer: implClose() completed");
873     }
874 
875     protected void implStart() {
876         if (Printer.trace) Printer.trace(">> RealTimeSequencer: implStart()");
877 
878         if (playThread == null) {
879             if (Printer.err) Printer.err("RealTimeSequencer.implStart() called, but playThread
                not instanciated!");
880             return;
881         }
882 
883         tempoCache.refresh(sequence);
884         if (!running) {
885             running  = true;
886             playThread.start();
887         }
888         if (Printer.trace) Printer.trace("<< RealTimeSequencer: implStart() completed");
889     }
890 
891 
892     protected void implStop() {
893         if (Printer.trace) Printer.trace(">> RealTimeSequencer: implStop()");
894 
895         if (playThread == null) {
896             if (Printer.err) Printer.err("RealTimeSequencer.implStop() called, but playThread not
                instanciated!");
897             return;
898         }
899 
900         recording = false;
901         if (running) {
902             running = false;
903             playThread.stop();
904         }
905         if (Printer.trace) Printer.trace("<< RealTimeSequencer: implStop() completed");
906     }
907 
908 
909     /**
910      * Send midi player events.
911      * must not be synchronized on "this"
912      */
913     protected void sendMetaEvents(MidiMessage message) {
914         if (metaEventListeners.size() == 0) return;
915 
916         //if (Printer.debug) Printer.debug("sending a meta event");
917         eventDispatcher.sendAudioEvents(message, metaEventListeners);
```

```
    }

    /**
     * Send midi player events.
     */
    protected void sendControllerEvents(MidiMessage message) {
        int size = controllerEventListeners.size();
        if (size == 0) return;

        //if (Printer.debug) Printer.debug("sending a controller event");

        if (! (message instanceof ShortMessage)) {
            if (Printer.debug) Printer.debug("sendControllerEvents: message is NOT instanceof
                ShortMessage!");
            return;
        }
        ShortMessage msg = (ShortMessage) message;
        int controller = msg.getData1();
        List sendToListeners = new ArrayList();
        for (int i = 0; i < size; i++) {
            ControllerListElement cve = (ControllerListElement) controllerEventListeners.get(i);
            for(int j = 0; j < cve.controllers.length; j++) {
                if (cve.controllers[j] == controller) {
                    sendToListeners.add(cve.listener);
                    break;
                }
            }
        }
        eventDispatcher.sendAudioEvents(message, sendToListeners);
    }



    private boolean needCaching() {
        return !isOpen() || (sequence == null) || (playThread == null);
    }

    /**
     * return the data pump instance, owned by play thread
     * if playthread is null, return null.
     * This method is guaranteed to return non-null if
     * needCaching returns false
     */
    private DataPump getDataPump() {
        if (playThread != null) {
            return playThread.getDataPump();
        }
        return null;
    }

    private MidiUtils.TempoCache getTempoCache() {
        return tempoCache;
    }

    private static boolean[] ensureBoolArraySize(boolean[] array, int desiredSize) {
        if (array == null) {
            return new boolean[desiredSize];
        }
        if (array.length < desiredSize) {
            boolean[] newArray = new boolean[desiredSize];
            System.arraycopy(array, 0, newArray, 0, array.length);
            return newArray;
```

```
        }
        return array;
    }


    // OVERRIDES OF ABSTRACT MIDI DEVICE METHODS

    protected boolean hasReceivers() {
        return true;
    }

    // for recording
    protected Receiver createReceiver() throws MidiUnavailableException {
        return new SequencerReceiver();
    }


    protected boolean hasTransmitters() {
        return true;
    }


    protected Transmitter createTransmitter() throws MidiUnavailableException {
        return new SequencerTransmitter();
    }


    // interface AutoConnectSequencer
    public void setAutoConnect(Receiver autoConnectedReceiver) {
        this.autoConnect = (autoConnectedReceiver != null);
        this.autoConnectedReceiver = autoConnectedReceiver;
    }



    // INNER CLASSES

    /**
     * An own class to distinguish the class name from
     * the transmitter of other devices
     */
    private class SequencerTransmitter extends BasicTransmitter {
        private SequencerTransmitter() {
            super();
        }
    }


    class SequencerReceiver extends AbstractReceiver {

        protected void implSend(MidiMessage message, long timeStamp) {
            if (recording) {
                long tickPos = 0;

                // convert timeStamp to ticks
                if (timeStamp < 0) {
                    tickPos = getTickPosition();
                } else {
                    synchronized(tempoCache) {
                        tickPos = MidiUtils.microsecond2tick(sequence, timeStamp, tempoCache);
                    }
                }
```

```
1041
1042                // and record to the first matching Track
1043                Track track = null;
1044                // do not record real-time events
1045                // see 5048381: NullPointerException when saving a MIDI sequence
1046                if (message.getLength() > 1) {
1047                    if (message instanceof ShortMessage) {
1048                        ShortMessage sm = (ShortMessage) message;
1049                        // all real-time messages have 0xF in the high nibble of the status byte
1050                        if ((sm.getStatus() & 0xF0) != 0xF0) {
1051                            track = RecordingTrack.get(recordingTracks, sm.getChannel());
1052                        }
1053                    } else {
1054                        // $$jb: where to record meta, sysex events?
1055                        // $$fb: the first recording track
1056                        track = RecordingTrack.get(recordingTracks, -1);
1057                    }
1058                    if (track != null) {
1059                        // create a copy of this message
1060                        if (message instanceof ShortMessage) {
1061                            message = new FastShortMessage((ShortMessage) message);
1062                        } else {
1063                            message = (MidiMessage) message.clone();
1064                        }
1065
1066                        // create new MidiEvent
1067                        MidiEvent me = new MidiEvent(message, tickPos);
1068                        track.add(me);
1069                    }
1070                }
1071            }
1072        }
1073    }
1074
1075
1076    private static class RealTimeSequencerInfo extends MidiDevice.Info {
1077
1078        private static final String name = "Real␣Time␣Sequencer";
1079        private static final String vendor = "Oracle␣Corporation";
1080        private static final String description = "Software␣sequencer";
1081        private static final String version = "Version␣1.0";
1082
1083        private RealTimeSequencerInfo() {
1084            super(name, vendor, description, version);
1085        }
1086    } // class Info
1087
1088
1089    private class ControllerListElement {
1090
1091        // $$jb: using an array for controllers b/c its
1092        //       easier to deal with than turning all the
1093        //       ints into objects to use a Vector
1094        int []  controllers;
1095        ControllerEventListener listener;
1096
1097        private ControllerListElement(ControllerEventListener listener, int[] controllers) {
1098
1099            this.listener = listener;
1100            if (controllers == null) {
1101                controllers = new int[128];
1102                for (int i = 0; i < 128; i++) {
```

```
                controllers[i] = i;
            }
        }
        this.controllers = controllers;
    }

    private void addControllers(int[] c) {

        if (c==null) {
            controllers = new int[128];
            for (int i = 0; i < 128; i++) {
                controllers[i] = i;
            }
            return;
        }
        int temp[] = new int[ controllers.length + c.length ];
        int elements;

        // first add what we have
        for(int i=0; i<controllers.length; i++) {
            temp[i] = controllers[i];
        }
        elements = controllers.length;
        // now add the new controllers only if we don't already have them
        for(int i=0; i<c.length; i++) {
            boolean flag = false;

            for(int j=0; j<controllers.length; j++) {
                if (c[i] == controllers[j]) {
                    flag = true;
                    break;
                }
            }
            if (!flag) {
                temp[elements++] = c[i];
            }
        }
        // now keep only the elements we need
        int newc[] = new int[ elements ];
        for(int i=0; i<elements; i++){
            newc[i] = temp[i];
        }
        controllers = newc;
    }

    private void removeControllers(int[] c) {

        if (c==null) {
            controllers = new int[0];
        } else {
            int temp[] = new int[ controllers.length ];
            int elements = 0;


            for(int i=0; i<controllers.length; i++){
                boolean flag = false;
                for(int j=0; j<c.length; j++) {
                    if (controllers[i] == c[j]) {
                        flag = true;
                        break;
                    }
                }
```

```
                if (!flag){
                    temp[elements++] = controllers[i];
                }
            }
            // now keep only the elements remaining
            int newc[] = new int[ elements ];
            for(int i=0; i<elements; i++) {
                newc[i] = temp[i];
            }
            controllers = newc;

        }
    }

    private int[] getControllers() {

        // return a copy of our array of controllers,
        // so others can't mess with it
        if (controllers == null) {
            return null;
        }

        int c[] = new int[controllers.length];

        for(int i=0; i<controllers.length; i++){
            c[i] = controllers[i];
        }
        return c;
    }

} // class ControllerListElement


static class RecordingTrack {

    private Track track;
    private int channel;

    RecordingTrack(Track track, int channel) {
        this.track = track;
        this.channel = channel;
    }

    static RecordingTrack get(List recordingTracks, Track track) {

        synchronized(recordingTracks) {
            int size = recordingTracks.size();

            for (int i = 0; i < size; i++) {
                RecordingTrack current = (RecordingTrack)recordingTracks.get(i);
                if (current.track == track) {
                    return current;
                }
            }
        }
        return null;
    }

    static Track get(List recordingTracks, int channel) {

        synchronized(recordingTracks) {
            int size = recordingTracks.size();
```

```
                for (int i = 0; i < size; i++) {
                    RecordingTrack current = (RecordingTrack)recordingTracks.get(i);
                    if ((current.channel == channel) || (current.channel == -1)) {
                        return current.track;
                    }
                }
            }
            return null;

        }
    }


    class PlayThread implements Runnable {
        private Thread thread;
        private Object lock = new Object();

        /** true if playback is interrupted (in close) */
        boolean interrupted = false;
        boolean isPumping = false;

        private DataPump dataPump = new DataPump();


        PlayThread() {
            // nearly MAX_PRIORITY
            int priority = Thread.NORM_PRIORITY
                + ((Thread.MAX_PRIORITY - Thread.NORM_PRIORITY) * 3) / 4;
            thread = JSSecurityManager.createThread(this,
                                                    "Java Sound Sequencer", // name
                                                    false,                  // daemon
                                                    priority,               // priority
                                                    true);                  // doStart
        }

        DataPump getDataPump() {
            return dataPump;
        }

        synchronized void setSequence(Sequence seq) {
            dataPump.setSequence(seq);
        }


        /** start thread and pump. Requires up-to-date tempoCache */
        synchronized void start() {
            // mark the sequencer running
            running = true;

            if (!dataPump.hasCachedTempo()) {
                long tickPos = getTickPosition();
                dataPump.setTempoMPQ(tempoCache.getTempoMPQAt(tickPos));
            }
            dataPump.checkPointMillis = 0; // means restarted
            dataPump.clearNoteOnCache();
            dataPump.needReindex = true;

            dataPump.resetLoopCount();

            // notify the thread
            synchronized(lock) {
                lock.notifyAll();
```

```
1289            }
1290
1291            if (Printer.debug) Printer.debug("␣->Started␣MIDI␣play␣thread");
1292
1293        }
1294
1295        // waits until stopped
1296        synchronized void stop() {
1297            playThreadImplStop();
1298            long t = System.nanoTime() / 1000000l;
1299            while (isPumping) {
1300                synchronized(lock) {
1301                    try {
1302                        lock.wait(2000);
1303                    } catch (InterruptedException ie) {
1304                        // ignore
1305                    }
1306                }
1307                // don't wait for more than 2 seconds
1308                if ((System.nanoTime()/1000000l) - t > 1900) {
1309                    if (Printer.err) Printer.err("Waited␣more␣than␣2␣seconds␣in␣RealTimeSequencer
                        .PlayThread.stop()!");
1310                    //break;
1311                }
1312            }
1313        }
1314
1315        void playThreadImplStop() {
1316            // mark the sequencer running
1317            running = false;
1318            synchronized(lock) {
1319                lock.notifyAll();
1320            }
1321        }
1322
1323        void close() {
1324            Thread oldThread = null;
1325            synchronized (this) {
1326                // dispose of thread
1327                interrupted = true;
1328                oldThread = thread;
1329                thread = null;
1330            }
1331            if (oldThread != null) {
1332                // wake up the thread if it's in wait()
1333                synchronized(lock) {
1334                    lock.notifyAll();
1335                }
1336            }
1337            // wait for the thread to terminate itself,
1338            // but max. 2 seconds. Must not be synchronized!
1339            if (oldThread != null) {
1340                try {
1341                    oldThread.join(2000);
1342                } catch (InterruptedException ie) {}
1343            }
1344        }
1345
1346
1347        /**
1348         * Main process loop driving the media flow.
1349         *
```

```
         * Make sure to NOT synchronize on RealTimeSequencer
         * anywhere here (even implicit). That is a sure deadlock!
         */
        public void run() {

            while (!interrupted) {
                boolean EOM = false;
                boolean wasRunning = running;
                isPumping = !interrupted && running;
                while (!EOM && !interrupted && running) {
                    EOM = dataPump.pump();

                    try {
                        Thread.sleep(1);
                    } catch (InterruptedException ie) {
                        // ignore
                    }
                }
                if (Printer.debug) {
                    Printer.debug("Exited main pump loop because: ");
                    if (EOM) Printer.debug(" -> EOM is reached");
                    if (!running) Printer.debug(" -> running was set to false");
                    if (interrupted) Printer.debug(" -> interrupted was set to true");
                }

                playThreadImplStop();
                if (wasRunning) {
                    dataPump.notesOff(true);
                }
                if (EOM) {
                    dataPump.setTickPos(sequence.getTickLength());

                    // send EOT event (mis-used for end of media)
                    MetaMessage message = new MetaMessage();
                    try{
                        message.setMessage(MidiUtils.META_END_OF_TRACK_TYPE, new byte[0], 0);
                    } catch(InvalidMidiDataException e1) {}
                    sendMetaEvents(message);
                }
                synchronized (lock) {
                    isPumping = false;
                    // wake up a waiting stop() method
                    lock.notifyAll();
                    while (!running && !interrupted) {
                        try {
                            lock.wait();
                        } catch (Exception ex) {}
                    }
                }
            } // end of while(!EOM && !interrupted && running)
            if (Printer.debug) Printer.debug("end of play thread");
        }
    }


    /**
     * class that does the actual dispatching of events,
     * used to be in native in MMAPI
     */
    private class DataPump {
        private float currTempo;         // MPQ tempo
        private float tempoFactor;       // 1.0 is default
```

```
1412        private float inverseTempoFactor;// = 1.0 / tempoFactor
1413        private long ignoreTempoEventAt; // ignore next META tempo during playback at this tick
                pos only
1414        private int resolution;
1415        private float divisionType;
1416        private long checkPointMillis;   // microseconds at checkoint
1417        private long checkPointTick;     // ticks at checkpoint
1418        private int[] noteOnCache;       // bit-mask of notes that are currently on
1419        private Track[] tracks;
1420        private boolean[] trackDisabled; // if true, do not play this track
1421        private int[] trackReadPos;      // read index per track
1422        private long lastTick;
1423        private boolean needReindex = false;
1424        private int currLoopCounter = 0;
1425
1426        //private sun.misc.Perf perf = sun.misc.Perf.getPerf();
1427        //private long perfFreq = perf.highResFrequency();
1428
1429
1430        DataPump() {
1431            init();
1432        }
1433
1434        synchronized void init() {
1435            ignoreTempoEventAt = -1;
1436            tempoFactor = 1.0f;
1437            inverseTempoFactor = 1.0f;
1438            noteOnCache = new int[128];
1439            tracks = null;
1440            trackDisabled = null;
1441        }
1442
1443        synchronized void setTickPos(long tickPos) {
1444            long oldLastTick = tickPos;
1445            lastTick = tickPos;
1446            if (running) {
1447                notesOff(false);
1448            }
1449            if (running || tickPos > 0) {
1450                // will also reindex
1451                chaseEvents(oldLastTick, tickPos);
1452            } else {
1453                needReindex = true;
1454            }
1455            if (!hasCachedTempo()) {
1456                setTempoMPQ(getTempoCache().getTempoMPQAt(lastTick, currTempo));
1457                // treat this as if it is a real time tempo change
1458                ignoreTempoEventAt = -1;
1459            }
1460            // trigger re-configuration
1461            checkPointMillis = 0;
1462        }
1463
1464        long getTickPos() {
1465            return lastTick;
1466        }
1467
1468        // hasCachedTempo is only valid if it is the current position
1469        boolean hasCachedTempo() {
1470            if (ignoreTempoEventAt != lastTick) {
1471                ignoreTempoEventAt = -1;
1472            }
```

```
            return ignoreTempoEventAt >= 0;
        }

        // this method is also used internally in the pump!
        synchronized void setTempoMPQ(float tempoMPQ) {
            if (tempoMPQ > 0 && tempoMPQ != currTempo) {
                ignoreTempoEventAt = lastTick;
                this.currTempo = tempoMPQ;
                // re-calculate check point
                checkPointMillis = 0;
            }
        }

        float getTempoMPQ() {
            return currTempo;
        }

        synchronized void setTempoFactor(float factor) {
            if (factor > 0 && factor != this.tempoFactor) {
                tempoFactor = factor;
                inverseTempoFactor = 1.0f / factor;
                // re-calculate check point
                checkPointMillis = 0;
            }
        }

        float getTempoFactor() {
            return tempoFactor;
        }

        synchronized void muteSoloChanged() {
            boolean[] newDisabled = makeDisabledArray();
            if (running) {
                applyDisabledTracks(trackDisabled, newDisabled);
            }
            trackDisabled = newDisabled;
        }



        synchronized void setSequence(Sequence seq) {
            if (seq == null) {
                init();
                return;
            }
            tracks = seq.getTracks();
            muteSoloChanged();
            resolution = seq.getResolution();
            divisionType = seq.getDivisionType();
            trackReadPos = new int[tracks.length];
            // trigger re-initialization
            checkPointMillis = 0;
            needReindex = true;
        }

        synchronized void resetLoopCount() {
            currLoopCounter = loopCount;
        }

        void clearNoteOnCache() {
            for (int i = 0; i < 128; i++) {
                noteOnCache[i] = 0;
```

```
1535            }
1536        }
1537
1538        void notesOff(boolean doControllers) {
1539            int done = 0;
1540            for (int ch=0; ch<16; ch++) {
1541                int channelMask = (1<<ch);
1542                for (int i=0; i<128; i++) {
1543                    if ((noteOnCache[i] & channelMask) != 0) {
1544                        noteOnCache[i] ^= channelMask;
1545                        // send note on with velocity 0
1546                        getTransmitterList().sendMessage((ShortMessage.NOTE_ON | ch) | (i<<8),
                                -1);
1547                        done++;
1548                    }
1549                }
1550                /* all notes off */
1551                getTransmitterList().sendMessage((ShortMessage.CONTROL_CHANGE | ch) | (123<<8),
                    -1);
1552                /* sustain off */
1553                getTransmitterList().sendMessage((ShortMessage.CONTROL_CHANGE | ch) | (64<<8),
                    -1);
1554                if (doControllers) {
1555                    /* reset all controllers */
1556                    getTransmitterList().sendMessage((ShortMessage.CONTROL_CHANGE | ch) |
                            (121<<8), -1);
1557                    done++;
1558                }
1559            }
1560            if (DEBUG_PUMP) Printer.println("  noteOff: sent "+done+" messages.");
1561        }
1562
1563
1564        private boolean[] makeDisabledArray() {
1565            if (tracks == null) {
1566                return null;
1567            }
1568            boolean[] newTrackDisabled = new boolean[tracks.length];
1569            boolean[] solo;
1570            boolean[] mute;
1571            synchronized(RealTimeSequencer.this) {
1572                mute = trackMuted;
1573                solo = trackSolo;
1574            }
1575            // if one track is solo, then only play solo
1576            boolean hasSolo = false;
1577            if (solo != null) {
1578                for (int i = 0; i < solo.length; i++) {
1579                    if (solo[i]) {
1580                        hasSolo = true;
1581                        break;
1582                    }
1583                }
1584            }
1585            if (hasSolo) {
1586                // only the channels with solo play, regardless of mute
1587                for (int i = 0; i < newTrackDisabled.length; i++) {
1588                    newTrackDisabled[i] = (i >= solo.length) || (!solo[i]);
1589                }
1590            } else {
1591                // mute the selected channels
1592                for (int i = 0; i < newTrackDisabled.length; i++) {
```

```
                newTrackDisabled[i] = (mute != null) && (i < mute.length) && (mute[i]);
            }
        }
        return newTrackDisabled;
    }

    /**
     * chase all events from beginning of Track
     * and send note off for those events that are active
     * in noteOnCache array.
     * It is possible, of course, to catch notes from other tracks,
     * but better than more complicated logic to detect
     * which notes are really from this track
     */
    private void sendNoteOffIfOn(Track track, long endTick) {
        int size = track.size();
        int done = 0;
        try {
            for (int i = 0; i < size; i++) {
                MidiEvent event = track.get(i);
                if (event.getTick() > endTick) break;
                MidiMessage msg = event.getMessage();
                int status = msg.getStatus();
                int len = msg.getLength();
                if (len == 3 && ((status & 0xF0) == ShortMessage.NOTE_ON)) {
                    int note = -1;
                    if (msg instanceof ShortMessage) {
                        ShortMessage smsg = (ShortMessage) msg;
                        if (smsg.getData2() > 0) {
                            // only consider Note On with velocity > 0
                            note = smsg.getData1();
                        }
                    } else {
                        byte[] data = msg.getMessage();
                        if ((data[2] & 0x7F) > 0) {
                            // only consider Note On with velocity > 0
                            note = data[1] & 0x7F;
                        }
                    }
                    if (note >= 0) {
                        int bit = 1<<(status & 0x0F);
                        if ((noteOnCache[note] & bit) != 0) {
                            // the bit is set. Send Note Off
                            getTransmitterList().sendMessage(status | (note<<8), -1);
                            // clear the bit
                            noteOnCache[note] &= (0xFFFF ^ bit);
                            done++;
                        }
                    }
                }
            }
        } catch (ArrayIndexOutOfBoundsException aioobe) {
            // this happens when messages are removed
            // from the track while this method executes
        }
        if (DEBUG_PUMP) Printer.println("  sendNoteOffIfOn: sent "+done+" messages.");
    }


    /**
     * Runtime application of mute/solo:
     * if a track is muted that was previously playing, send
```

```
1655          *    note off events for all currently playing notes
1656          */
1657         private void applyDisabledTracks(boolean[] oldDisabled, boolean[] newDisabled) {
1658             byte[][] tempArray = null;
1659             synchronized(RealTimeSequencer.this) {
1660                 for (int i = 0; i < newDisabled.length; i++) {
1661                     if (((oldDisabled == null)
1662                          || (i >= oldDisabled.length)
1663                          || !oldDisabled[i])
1664                         && newDisabled[i]) {
1665                         // case that a track gets muted: need to
1666                         // send appropriate note off events to prevent
1667                         // hanging notes
1668 
1669                         if (tracks.length > i) {
1670                             sendNoteOffIfOn(tracks[i], lastTick);
1671                         }
1672                     }
1673                     else if ((oldDisabled != null)
1674                              && (i < oldDisabled.length)
1675                              && oldDisabled[i]
1676                              && !newDisabled[i]) {
1677                         // case that a track was muted and is now unmuted
1678                         // need to chase events and re-index this track
1679                         if (tempArray == null) {
1680                             tempArray = new byte[128][16];
1681                         }
1682                         chaseTrackEvents(i, 0, lastTick, true, tempArray);
1683                     }
1684                 }
1685             }
1686         }
1687 
1688         /** go through all events from startTick to endTick
1689          * chase the controller state and program change state
1690          * and then set the end-states at once.
1691          *
1692          * needs to be called in synchronized state
1693          * @param tempArray an byte[128][16] to hold controller messages
1694          */
1695         private void chaseTrackEvents(int trackNum,
1696                                       long startTick,
1697                                       long endTick,
1698                                       boolean doReindex,
1699                                       byte[][] tempArray) {
1700             if (startTick > endTick) {
1701                 // start from the beginning
1702                 startTick = 0;
1703             }
1704             byte[] progs = new byte[16];
1705             // init temp array with impossible values
1706             for (int ch = 0; ch < 16; ch++) {
1707                 progs[ch] = -1;
1708                 for (int co = 0; co < 128; co++) {
1709                     tempArray[co][ch] = -1;
1710                 }
1711             }
1712             Track track = tracks[trackNum];
1713             int size = track.size();
1714             try {
1715                 for (int i = 0; i < size; i++) {
1716                     MidiEvent event = track.get(i);
```

```
                if (event.getTick() >= endTick) {
                    if (doReindex && (trackNum < trackReadPos.length)) {
                        trackReadPos[trackNum] = (i > 0)?(i-1):0;
                        if (DEBUG_PUMP) Printer.println("  chaseEvents: setting trackReadPos[
                            "+trackNum+"] = "+trackReadPos[trackNum]);
                    }
                    break;
                }
                MidiMessage msg = event.getMessage();
                int status = msg.getStatus();
                int len = msg.getLength();
                if (len == 3 && ((status & 0xF0) == ShortMessage.CONTROL_CHANGE)) {
                    if (msg instanceof ShortMessage) {
                        ShortMessage smsg = (ShortMessage) msg;
                        tempArray[smsg.getData1() & 0x7F][status & 0x0F] = (byte) smsg.
                            getData2();
                    } else {
                        byte[] data = msg.getMessage();
                        tempArray[data[1] & 0x7F][status & 0x0F] = data[2];
                    }
                }
                if (len == 2 && ((status & 0xF0) == ShortMessage.PROGRAM_CHANGE)) {
                    if (msg instanceof ShortMessage) {
                        ShortMessage smsg = (ShortMessage) msg;
                        progs[status & 0x0F] = (byte) smsg.getData1();
                    } else {
                        byte[] data = msg.getMessage();
                        progs[status & 0x0F] = data[1];
                    }
                }
            }
        } catch (ArrayIndexOutOfBoundsException aioobe) {
            // this happens when messages are removed
            // from the track while this method executes
        }
        int numControllersSent = 0;
        // now send out the aggregated controllers and program changes
        for (int ch = 0; ch < 16; ch++) {
            for (int co = 0; co < 128; co++) {
                byte controllerValue = tempArray[co][ch];
                if (controllerValue >= 0) {
                    int packedMsg = (ShortMessage.CONTROL_CHANGE | ch) | (co<<8) | (
                        controllerValue<<16);
                    getTransmitterList().sendMessage(packedMsg, -1);
                    numControllersSent++;
                }
            }
            // send program change *after* controllers, to
            // correctly initialize banks
            if (progs[ch] >= 0) {
                getTransmitterList().sendMessage((ShortMessage.PROGRAM_CHANGE | ch) | (progs[
                    ch]<<8), -1);
            }
            if (progs[ch] >= 0 || startTick == 0 || endTick == 0) {
                // reset pitch bend on this channel (E0 00 40)
                getTransmitterList().sendMessage((ShortMessage.PITCH_BEND | ch) | (0x40 <<
                    16), -1);
                // reset sustain pedal on this channel
                getTransmitterList().sendMessage((ShortMessage.CONTROL_CHANGE | ch) | (64 <<
                    8), -1);
            }
        }
```

```
1773            if (DEBUG_PUMP) Printer.println("  chaseTrackEvents track "+trackNum+": sent "+
                    numControllersSent+" controllers.");
1774        }
1775
1776
1777        /** chase controllers and program for all tracks */
1778        synchronized void chaseEvents(long startTick, long endTick) {
1779            if (DEBUG_PUMP) Printer.println(">> chaseEvents from tick "+startTick+".."+(endTick
                    -1));
1780            byte[][] tempArray = new byte[128][16];
1781            for (int t = 0; t < tracks.length; t++) {
1782                if ((trackDisabled == null)
1783                    || (trackDisabled.length <= t)
1784                    || (!trackDisabled[t])) {
1785                    // if track is not disabled, chase the events for it
1786                    chaseTrackEvents(t, startTick, endTick, true, tempArray);
1787                }
1788            }
1789            if (DEBUG_PUMP) Printer.println("<< chaseEvents");
1790        }
1791
1792
1793        // playback related methods (pumping)
1794
1795        private long getCurrentTimeMillis() {
1796            return System.nanoTime() / 1000000l;
1797            //return perf.highResCounter() * 1000 / perfFreq;
1798        }
1799
1800        private long millis2tick(long millis) {
1801            if (divisionType != Sequence.PPQ) {
1802                double dTick = ((((double) millis) * tempoFactor)
1803                                * ((double) divisionType)
1804                                * ((double) resolution))
1805                    / ((double) 1000);
1806                return (long) dTick;
1807            }
1808            return MidiUtils.microsec2ticks(millis * 1000,
1809                                            currTempo * inverseTempoFactor,
1810                                            resolution);
1811        }
1812
1813        private long tick2millis(long tick) {
1814            if (divisionType != Sequence.PPQ) {
1815                double dMillis = ((((double) tick) * 1000) /
1816                                  (tempoFactor * ((double) divisionType) * ((double) resolution))
                                      );
1817                return (long) dMillis;
1818            }
1819            return MidiUtils.ticks2microsec(tick,
1820                                            currTempo * inverseTempoFactor,
1821                                            resolution) / 1000;
1822        }
1823
1824        private void ReindexTrack(int trackNum, long tick) {
1825            if (trackNum < trackReadPos.length && trackNum < tracks.length) {
1826                trackReadPos[trackNum] = MidiUtils.tick2index(tracks[trackNum], tick);
1827                if (DEBUG_PUMP) Printer.println("  reindexTrack: setting trackReadPos["+trackNum+
                        "] = "+trackReadPos[trackNum]);
1828            }
1829        }
1830
```

```
        /* returns if changes are pending */
        private boolean dispatchMessage(int trackNum, MidiEvent event) {
            boolean changesPending = false;
            MidiMessage message = event.getMessage();
            int msgStatus = message.getStatus();
            int msgLen = message.getLength();
            if (msgStatus == MetaMessage.META && msgLen >= 2) {
                // a meta message. Do not send it to the device.
                // 0xFF with length=1 is a MIDI realtime message
                // which shouldn't be in a Sequence, but we play it
                // nonetheless.

                // see if this is a tempo message. Only on track 0.
                if (trackNum == 0) {
                    int newTempo = MidiUtils.getTempoMPQ(message);
                    if (newTempo > 0) {
                        if (event.getTick() != ignoreTempoEventAt) {
                            setTempoMPQ(newTempo); // sets ignoreTempoEventAt!
                            changesPending = true;
                        }
                        // next loop, do not ignore anymore tempo events.
                        ignoreTempoEventAt = -1;
                    }
                }
                // send to listeners
                sendMetaEvents(message);

            } else {
                // not meta, send to device
                getTransmitterList().sendMessage(message, -1);

                switch (msgStatus & 0xF0) {
                case ShortMessage.NOTE_OFF: {
                    // note off - clear the bit in the noteOnCache array
                    int note = ((ShortMessage) message).getData1() & 0x7F;
                    noteOnCache[note] &= (0xFFFF ^ (1<<(msgStatus & 0x0F)));
                    break;
                }

                case ShortMessage.NOTE_ON: {
                    // note on
                    ShortMessage smsg = (ShortMessage) message;
                    int note = smsg.getData1() & 0x7F;
                    int vel = smsg.getData2() & 0x7F;
                    if (vel > 0) {
                        // if velocity > 0 set the bit in the noteOnCache array
                        noteOnCache[note] |= 1<<(msgStatus & 0x0F);
                    } else {
                        // if velocity = 0 clear the bit in the noteOnCache array
                        noteOnCache[note] &= (0xFFFF ^ (1<<(msgStatus & 0x0F)));
                    }
                    break;
                }

                case ShortMessage.CONTROL_CHANGE:
                    // if controller message, send controller listeners
                    sendControllerEvents(message);
                    break;

                }
            }
            return changesPending;
```

```
1893        }
1894
1895
1896        /** the main pump method
1897         * @return true if end of sequence is reached
1898         */
1899        synchronized boolean pump() {
1900            long currMillis;
1901            long targetTick = lastTick;
1902            MidiEvent currEvent;
1903            boolean changesPending = false;
1904            boolean doLoop = false;
1905            boolean EOM = false;
1906
1907            currMillis = getCurrentTimeMillis();
1908            int finishedTracks = 0;
1909            do {
1910                changesPending = false;
1911
1912                // need to re-find indexes in tracks?
1913                if (needReindex) {
1914                    if (DEBUG_PUMP) Printer.println("Need␣to␣re-index␣at␣"+currMillis+"␣millis.␣
                           TargetTick="+targetTick);
1915                    if (trackReadPos.length < tracks.length) {
1916                        trackReadPos = new int[tracks.length];
1917                    }
1918                    for (int t = 0; t < tracks.length; t++) {
1919                        ReindexTrack(t, targetTick);
1920                        if (DEBUG_PUMP_ALL) Printer.println("␣␣Setting␣trackReadPos["+t+"]="+
                               trackReadPos[t]);
1921                    }
1922                    needReindex = false;
1923                    checkPointMillis = 0;
1924                }
1925
1926                // get target tick from current time in millis
1927                if (checkPointMillis == 0) {
1928                    // new check point
1929                    currMillis = getCurrentTimeMillis();
1930                    checkPointMillis = currMillis;
1931                    targetTick = lastTick;
1932                    checkPointTick = targetTick;
1933                    if (DEBUG_PUMP) Printer.println("New␣checkpoint␣to␣"+currMillis+"␣millis.␣"
1934                                                    +"TargetTick="+targetTick
1935                                                    +"␣new␣tempo="+MidiUtils.convertTempo(
                                                           currTempo)+"bpm");
1936                } else {
1937                    // calculate current tick based on current time in milliseconds
1938                    targetTick = checkPointTick + millis2tick(currMillis - checkPointMillis);
1939                    if (DEBUG_PUMP_ALL) Printer.println("targetTick␣=␣"+targetTick+"␣at␣"+
                           currMillis+"␣millis");
1940                    if ((loopEnd != -1)
1941                        && ((loopCount > 0 && currLoopCounter > 0)
1942                            || (loopCount == LOOP_CONTINUOUSLY))) {
1943                        if (lastTick <= loopEnd && targetTick >= loopEnd) {
1944                            // need to loop!
1945                            // only play until loop end
1946                            targetTick = loopEnd - 1;
1947                            doLoop = true;
1948                            if (DEBUG_PUMP) Printer.println("set␣doLoop␣to␣true.␣lastTick="+
                                   lastTick
1949                                                            +"␣␣targetTick="+targetTick
```

```
                                                    +"  loopEnd="+loopEnd
                                                    +"  jumping to loopStart="+
                                                        loopStart
                                                    +"  new currLoopCounter="+
                                                        currLoopCounter);
                    if (DEBUG_PUMP) Printer.println("  currMillis="+currMillis
                                                    +"  checkPointMillis="+
                                                        checkPointMillis
                                                    +"  checkPointTick="+
                                                        checkPointTick);

                }
            }
            lastTick = targetTick;
        }

        finishedTracks = 0;

        for (int t = 0; t < tracks.length; t++) {
            try {
                boolean disabled = trackDisabled[t];
                Track thisTrack = tracks[t];
                int readPos = trackReadPos[t];
                int size = thisTrack.size();
                // play all events that are due until targetTick
                while (!changesPending && (readPos < size)
                       && (currEvent = thisTrack.get(readPos)).getTick() <= targetTick) {

                    if ((readPos == size -1) &&  MidiUtils.isMetaEndOfTrack(currEvent.
                        getMessage())) {
                        // do not send out this message. Finished with this track
                        readPos = size;
                        break;
                    }
                    // TODO: some kind of heuristics if the MIDI messages have changed
                    // significantly (i.e. deleted or inserted a bunch of messages)
                    // since last time. Would need to set needReindex = true then
                    readPos++;
                    // only play this event if the track is enabled,
                    // or if it is a tempo message on track 0
                    // Note: cannot put this check outside
                    //       this inner loop in order to detect end of file
                    if (!disabled ||
                        ((t == 0) && (MidiUtils.isMetaTempo(currEvent.getMessage())))) {
                        changesPending = dispatchMessage(t, currEvent);
                    }
                }
                if (readPos >= size) {
                    finishedTracks++;
                }
                if (DEBUG_PUMP_ALL) {
                    System.out.print(" pumped track "+t+" ("+size+" events) "
                                     +" from index: "+trackReadPos[t]
                                     +" to "+(readPos-1));
                    System.out.print(" -> ticks: ");
                    if (trackReadPos[t] < size) {
                        System.out.print(""+(thisTrack.get(trackReadPos[t]).getTick()));
                    } else {
                        System.out.print("EOT");
                    }
                    System.out.print(" to ");
                    if (readPos < size) {
```

```
                        System.out.print(""+(thisTrack.get(readPos-1).getTick()));
                    } else {
                        System.out.print("EOT");
                    }
                    System.out.println();
                }
                trackReadPos[t] = readPos;
            } catch(Exception e) {
                if (Printer.debug) Printer.debug("Exception␣in␣Sequencer␣pump!");
                if (Printer.debug) e.printStackTrace();
                if (e instanceof ArrayIndexOutOfBoundsException) {
                    needReindex = true;
                    changesPending = true;
                }
            }
            if (changesPending) {
                break;
            }
        }
        EOM = (finishedTracks == tracks.length);
        if (doLoop
            || ( ((loopCount > 0 && currLoopCounter > 0)
                  || (loopCount == LOOP_CONTINUOUSLY))
                 && !changesPending
                 && (loopEnd == -1)
                 && EOM)) {

            long oldCheckPointMillis = checkPointMillis;
            long loopEndTick = loopEnd;
            if (loopEndTick == -1) {
                loopEndTick = lastTick;
            }

            // need to loop back!
            if (loopCount != LOOP_CONTINUOUSLY) {
                currLoopCounter--;
            }
            if (DEBUG_PUMP) Printer.println("Execute␣loop:␣lastTick="+lastTick
                                            +"␣␣loopEnd="+loopEnd
                                            +"␣␣jumping␣to␣loopStart="+loopStart
                                            +"␣␣new␣currLoopCounter="+currLoopCounter)
                                                ;
            setTickPos(loopStart);
            // now patch the checkPointMillis so that
            // it points to the exact beginning of when the loop was finished

            // $$fb TODO: although this is mathematically correct (i.e. the loop position
            //            is correct, and doesn't drift away with several repetition,
            //            there is a slight lag when looping back, probably caused
            //            by the chasing.

            checkPointMillis = oldCheckPointMillis + tick2millis(loopEndTick -
                checkPointTick);
            checkPointTick = loopStart;
            if (DEBUG_PUMP) Printer.println("␣␣Setting␣currMillis="+currMillis
                                            +"␣␣new␣checkPointMillis="+
                                                checkPointMillis
                                            +"␣␣new␣checkPointTick="+checkPointTick);
            // no need for reindexing, is done in setTickPos
            needReindex = false;
            changesPending = false;
            // reset doLoop flag
```

```
                    doLoop = false;
                    EOM = false;
                }
            } while (changesPending);

            return EOM;
        }

    } // class DataPump

}
```

65 com/sun/media/sound/SF2GlobalRegion.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * Soundfont global region.
 *
 * @author Karl Helgason
 */
public class SF2GlobalRegion extends SF2Region {
}
```

66 com/sun/media/sound/SF2Instrument.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import javax.sound.midi.Patch;

/**
 * Soundfont instrument.
 *
 * @author Karl Helgason
 */
public class SF2Instrument extends ModelInstrument {

    protected String name = "";
    protected int preset = 0;
    protected int bank = 0;
    protected long library = 0;
    protected long genre = 0;
    protected long morphology = 0;
    protected SF2GlobalRegion globalregion = null;
    protected List<SF2InstrumentRegion> regions
            = new ArrayList<SF2InstrumentRegion>();

    public SF2Instrument() {
        super(null, null, null, null);
    }

    public SF2Instrument(SF2Soundbank soundbank) {
        super(soundbank, null, null, null);
    }

    public String getName() {
        return name;
```

```
    }

    public void setName(String name) {
        this.name = name;
    }

    public Patch getPatch() {
        if (bank == 128)
            return new ModelPatch(0, preset, true);
        else
            return new ModelPatch(bank << 7, preset, false);
    }

    public void setPatch(Patch patch) {
        if (patch instanceof ModelPatch && ((ModelPatch) patch).isPercussion()) {
            bank = 128;
            preset = patch.getProgram();
        } else {
            bank = patch.getBank() >> 7;
            preset = patch.getProgram();
        }
    }

    public Object getData() {
        return null;
    }

    public long getGenre() {
        return genre;
    }

    public void setGenre(long genre) {
        this.genre = genre;
    }

    public long getLibrary() {
        return library;
    }

    public void setLibrary(long library) {
        this.library = library;
    }

    public long getMorphology() {
        return morphology;
    }

    public void setMorphology(long morphology) {
        this.morphology = morphology;
    }

    public List<SF2InstrumentRegion> getRegions() {
        return regions;
    }

    public SF2GlobalRegion getGlobalRegion() {
        return globalregion;
    }

    public void setGlobalZone(SF2GlobalRegion zone) {
        globalregion = zone;
    }
```

```
123
124     public String toString() {
125         if (bank == 128)
126             return "Drumkit: " + name + " preset #" + preset;
127         else
128             return "Instrument: " + name + " bank #" + bank
129                     + " preset #" + preset;
130     }
131
132     public ModelPerformer[] getPerformers() {
133         int performercount = 0;
134         for (SF2InstrumentRegion presetzone : regions)
135             performercount += presetzone.getLayer().getRegions().size();
136         ModelPerformer[] performers = new ModelPerformer[performercount];
137         int pi = 0;
138
139         SF2GlobalRegion presetglobal = globalregion;
140         for (SF2InstrumentRegion presetzone : regions) {
141             Map<Integer, Short> pgenerators = new HashMap<Integer, Short>();
142             pgenerators.putAll(presetzone.getGenerators());
143             if (presetglobal != null)
144                 pgenerators.putAll(presetglobal.getGenerators());
145
146             SF2Layer layer = presetzone.getLayer();
147             SF2GlobalRegion layerglobal = layer.getGlobalRegion();
148             for (SF2LayerRegion layerzone : layer.getRegions()) {
149                 ModelPerformer performer = new ModelPerformer();
150                 if (layerzone.getSample() != null)
151                     performer.setName(layerzone.getSample().getName());
152                 else
153                     performer.setName(layer.getName());
154
155                 performers[pi++] = performer;
156
157                 int keyfrom = 0;
158                 int keyto = 127;
159                 int velfrom = 0;
160                 int velto = 127;
161
162                 if (layerzone.contains(SF2Region.GENERATOR_EXCLUSIVECLASS)) {
163                     performer.setExclusiveClass(layerzone.getInteger(
164                             SF2Region.GENERATOR_EXCLUSIVECLASS));
165                 }
166                 if (layerzone.contains(SF2Region.GENERATOR_KEYRANGE)) {
167                     byte[] bytes = layerzone.getBytes(
168                             SF2Region.GENERATOR_KEYRANGE);
169                     if (bytes[0] >= 0)
170                         if (bytes[0] > keyfrom)
171                             keyfrom = bytes[0];
172                     if (bytes[1] >= 0)
173                         if (bytes[1] < keyto)
174                             keyto = bytes[1];
175                 }
176                 if (layerzone.contains(SF2Region.GENERATOR_VELRANGE)) {
177                     byte[] bytes = layerzone.getBytes(
178                             SF2Region.GENERATOR_VELRANGE);
179                     if (bytes[0] >= 0)
180                         if (bytes[0] > velfrom)
181                             velfrom = bytes[0];
182                     if (bytes[1] >= 0)
183                         if (bytes[1] < velto)
184                             velto = bytes[1];
```

```
                }
                if (presetzone.contains(SF2Region.GENERATOR_KEYRANGE)) {
                    byte[] bytes = presetzone.getBytes(
                            SF2Region.GENERATOR_KEYRANGE);
                    if (bytes[0] > keyfrom)
                        keyfrom = bytes[0];
                    if (bytes[1] < keyto)
                        keyto = bytes[1];
                }
                if (presetzone.contains(SF2Region.GENERATOR_VELRANGE)) {
                    byte[] bytes = presetzone.getBytes(
                            SF2Region.GENERATOR_VELRANGE);
                    if (bytes[0] > velfrom)
                        velfrom = bytes[0];
                    if (bytes[1] < velto)
                        velto = bytes[1];
                }
                performer.setKeyFrom(keyfrom);
                performer.setKeyTo(keyto);
                performer.setVelFrom(velfrom);
                performer.setVelTo(velto);

                int startAddrsOffset = layerzone.getShort(
                        SF2Region.GENERATOR_STARTADDRSOFFSET);
                int endAddrsOffset = layerzone.getShort(
                        SF2Region.GENERATOR_ENDADDRSOFFSET);
                int startloopAddrsOffset = layerzone.getShort(
                        SF2Region.GENERATOR_STARTLOOPADDRSOFFSET);
                int endloopAddrsOffset = layerzone.getShort(
                        SF2Region.GENERATOR_ENDLOOPADDRSOFFSET);

                startAddrsOffset += layerzone.getShort(
                        SF2Region.GENERATOR_STARTADDRSCOARSEOFFSET) * 32768;
                endAddrsOffset += layerzone.getShort(
                        SF2Region.GENERATOR_ENDADDRSCOARSEOFFSET) * 32768;
                startloopAddrsOffset += layerzone.getShort(
                        SF2Region.GENERATOR_STARTLOOPADDRSCOARSEOFFSET) * 32768;
                endloopAddrsOffset += layerzone.getShort(
                        SF2Region.GENERATOR_ENDLOOPADDRSCOARSEOFFSET) * 32768;
                startloopAddrsOffset -= startAddrsOffset;
                endloopAddrsOffset -= startAddrsOffset;

                SF2Sample sample = layerzone.getSample();
                int rootkey = sample.originalPitch;
                if (layerzone.getShort(SF2Region.GENERATOR_OVERRIDINGROOTKEY) != -1) {
                    rootkey = layerzone.getShort(
                            SF2Region.GENERATOR_OVERRIDINGROOTKEY);
                }
                float pitchcorrection = (-rootkey * 100) + sample.pitchCorrection;
                ModelByteBuffer buff = sample.getDataBuffer();
                ModelByteBuffer buff24 = sample.getData24Buffer();

                if (startAddrsOffset != 0 || endAddrsOffset != 0) {
                    buff = buff.subbuffer(startAddrsOffset * 2,
                            buff.capacity() + endAddrsOffset * 2);
                    if (buff24 != null) {
                        buff24 = buff24.subbuffer(startAddrsOffset,
                                buff24.capacity() + endAddrsOffset);
                    }

                    /*
                    if (startAddrsOffset < 0)
```

```
                startAddrsOffset = 0;
            if (endAddrsOffset > (buff.capacity()/2-startAddrsOffset))
                startAddrsOffset = (int)buff.capacity()/2-startAddrsOffset;
            byte[] data = buff.array();
            int off = (int)buff.arrayOffset() + startAddrsOffset*2;
            int len = (int)buff.capacity() + endAddrsOffset*2;
            if (off+len > data.length)
                len = data.length - off;
            buff = new ModelByteBuffer(data, off, len);
            if(buff24 != null) {
                data = buff.array();
                off = (int)buff.arrayOffset() + startAddrsOffset;
                len = (int)buff.capacity() + endAddrsOffset;
                buff24 = new ModelByteBuffer(data, off, len);
            }
            */
        }

        ModelByteBufferWavetable osc = new ModelByteBufferWavetable(
                buff, sample.getFormat(), pitchcorrection);
        if (buff24 != null)
            osc.set8BitExtensionBuffer(buff24);

        Map<Integer, Short> generators = new HashMap<Integer, Short>();
        if (layerglobal != null)
            generators.putAll(layerglobal.getGenerators());
        generators.putAll(layerzone.getGenerators());
        for (Map.Entry<Integer, Short> gen : pgenerators.entrySet()) {
            short val;
            if (!generators.containsKey(gen.getKey()))
                val = layerzone.getShort(gen.getKey());
            else
                val = generators.get(gen.getKey());
            val += gen.getValue();
            generators.put(gen.getKey(), val);
        }

        // SampleMode:
        // 0 indicates a sound reproduced with no loop
        // 1 indicates a sound which loops continuously
        // 2 is unused but should be interpreted as indicating no loop
        // 3 indicates a sound which loops for the duration of key
        //   depression then proceeds to play the remainder of the sample.
        int sampleMode = getGeneratorValue(generators,
                SF2Region.GENERATOR_SAMPLEMODES);
        if ((sampleMode == 1) || (sampleMode == 3)) {
            if (sample.startLoop >= 0 && sample.endLoop > 0) {
                osc.setLoopStart((int)(sample.startLoop
                        + startloopAddrsOffset));
                osc.setLoopLength((int)(sample.endLoop - sample.startLoop
                        + endloopAddrsOffset - startloopAddrsOffset));
                if (sampleMode == 1)
                    osc.setLoopType(ModelWavetable.LOOP_TYPE_FORWARD);
                if (sampleMode == 3)
                    osc.setLoopType(ModelWavetable.LOOP_TYPE_RELEASE);
            }
        }
        performer.getOscillators().add(osc);


        short volDelay = getGeneratorValue(generators,
                SF2Region.GENERATOR_DELAYVOLENV);
```

```
309                 short volAttack = getGeneratorValue(generators,
310                         SF2Region.GENERATOR_ATTACKVOLENV);
311                 short volHold = getGeneratorValue(generators,
312                         SF2Region.GENERATOR_HOLDVOLENV);
313                 short volDecay = getGeneratorValue(generators,
314                         SF2Region.GENERATOR_DECAYVOLENV);
315                 short volSustain = getGeneratorValue(generators,
316                         SF2Region.GENERATOR_SUSTAINVOLENV);
317                 short volRelease = getGeneratorValue(generators,
318                         SF2Region.GENERATOR_RELEASEVOLENV);
319
320                 if (volHold != -12000) {
321                     short volKeyNumToHold = getGeneratorValue(generators,
322                             SF2Region.GENERATOR_KEYNUMTOVOLENVHOLD);
323                     volHold += 60 * volKeyNumToHold;
324                     float fvalue = -volKeyNumToHold * 128;
325                     ModelIdentifier src = ModelSource.SOURCE_NOTEON_KEYNUMBER;
326                     ModelIdentifier dest = ModelDestination.DESTINATION_EG1_HOLD;
327                     performer.getConnectionBlocks().add(
328                         new ModelConnectionBlock(new ModelSource(src), fvalue,
329                             new ModelDestination(dest)));
330                 }
331                 if (volDecay != -12000) {
332                     short volKeyNumToDecay = getGeneratorValue(generators,
333                             SF2Region.GENERATOR_KEYNUMTOVOLENVDECAY);
334                     volDecay += 60 * volKeyNumToDecay;
335                     float fvalue = -volKeyNumToDecay * 128;
336                     ModelIdentifier src = ModelSource.SOURCE_NOTEON_KEYNUMBER;
337                     ModelIdentifier dest = ModelDestination.DESTINATION_EG1_DECAY;
338                     performer.getConnectionBlocks().add(
339                         new ModelConnectionBlock(new ModelSource(src), fvalue,
340                             new ModelDestination(dest)));
341                 }
342
343                 addTimecentValue(performer,
344                         ModelDestination.DESTINATION_EG1_DELAY, volDelay);
345                 addTimecentValue(performer,
346                         ModelDestination.DESTINATION_EG1_ATTACK, volAttack);
347                 addTimecentValue(performer,
348                         ModelDestination.DESTINATION_EG1_HOLD, volHold);
349                 addTimecentValue(performer,
350                         ModelDestination.DESTINATION_EG1_DECAY, volDecay);
351                 //float fvolsustain = (960-volSustain)*(1000.0f/960.0f);
352
353                 volSustain = (short)(1000 - volSustain);
354                 if (volSustain < 0)
355                     volSustain = 0;
356                 if (volSustain > 1000)
357                     volSustain = 1000;
358
359                 addValue(performer,
360                         ModelDestination.DESTINATION_EG1_SUSTAIN, volSustain);
361                 addTimecentValue(performer,
362                         ModelDestination.DESTINATION_EG1_RELEASE, volRelease);
363
364                 if (getGeneratorValue(generators,
365                             SF2Region.GENERATOR_MODENVTOFILTERFC) != 0
366                         || getGeneratorValue(generators,
367                             SF2Region.GENERATOR_MODENVTOPITCH) != 0) {
368                     short modDelay = getGeneratorValue(generators,
369                             SF2Region.GENERATOR_DELAYMODENV);
370                     short modAttack = getGeneratorValue(generators,
```

```
                SF2Region.GENERATOR_ATTACKMODENV);
        short modHold = getGeneratorValue(generators,
                SF2Region.GENERATOR_HOLDMODENV);
        short modDecay = getGeneratorValue(generators,
                SF2Region.GENERATOR_DECAYMODENV);
        short modSustain = getGeneratorValue(generators,
                SF2Region.GENERATOR_SUSTAINMODENV);
        short modRelease = getGeneratorValue(generators,
                SF2Region.GENERATOR_RELEASEMODENV);


        if (modHold != -12000) {
            short modKeyNumToHold = getGeneratorValue(generators,
                    SF2Region.GENERATOR_KEYNUMTOMODENVHOLD);
            modHold += 60 * modKeyNumToHold;
            float fvalue = -modKeyNumToHold * 128;
            ModelIdentifier src = ModelSource.SOURCE_NOTEON_KEYNUMBER;
            ModelIdentifier dest = ModelDestination.DESTINATION_EG2_HOLD;
            performer.getConnectionBlocks().add(
                new ModelConnectionBlock(new ModelSource(src),
                    fvalue, new ModelDestination(dest)));
        }
        if (modDecay != -12000) {
            short modKeyNumToDecay = getGeneratorValue(generators,
                    SF2Region.GENERATOR_KEYNUMTOMODENVDECAY);
            modDecay += 60 * modKeyNumToDecay;
            float fvalue = -modKeyNumToDecay * 128;
            ModelIdentifier src = ModelSource.SOURCE_NOTEON_KEYNUMBER;
            ModelIdentifier dest = ModelDestination.DESTINATION_EG2_DECAY;
            performer.getConnectionBlocks().add(
                new ModelConnectionBlock(new ModelSource(src),
                    fvalue, new ModelDestination(dest)));
        }

        addTimecentValue(performer,
                ModelDestination.DESTINATION_EG2_DELAY, modDelay);
        addTimecentValue(performer,
                ModelDestination.DESTINATION_EG2_ATTACK, modAttack);
        addTimecentValue(performer,
                ModelDestination.DESTINATION_EG2_HOLD, modHold);
        addTimecentValue(performer,
                ModelDestination.DESTINATION_EG2_DECAY, modDecay);
        if (modSustain < 0)
            modSustain = 0;
        if (modSustain > 1000)
            modSustain = 1000;
        addValue(performer, ModelDestination.DESTINATION_EG2_SUSTAIN,
                1000 - modSustain);
        addTimecentValue(performer,
                ModelDestination.DESTINATION_EG2_RELEASE, modRelease);

        if (getGeneratorValue(generators,
                SF2Region.GENERATOR_MODENVTOFILTERFC) != 0) {
            double fvalue = getGeneratorValue(generators,
                    SF2Region.GENERATOR_MODENVTOFILTERFC);
            ModelIdentifier src = ModelSource.SOURCE_EG2;
            ModelIdentifier dest
                    = ModelDestination.DESTINATION_FILTER_FREQ;
            performer.getConnectionBlocks().add(
                new ModelConnectionBlock(new ModelSource(src),
                    fvalue, new ModelDestination(dest)));
        }
```

```
433
434                     if (getGeneratorValue(generators,
435                             SF2Region.GENERATOR_MODENVTOPITCH) != 0) {
436                         double fvalue = getGeneratorValue(generators,
437                                 SF2Region.GENERATOR_MODENVTOPITCH);
438                         ModelIdentifier src = ModelSource.SOURCE_EG2;
439                         ModelIdentifier dest = ModelDestination.DESTINATION_PITCH;
440                         performer.getConnectionBlocks().add(
441                             new ModelConnectionBlock(new ModelSource(src),
442                                 fvalue, new ModelDestination(dest)));
443                     }
444
445                 }
446
447                 if (getGeneratorValue(generators,
448                             SF2Region.GENERATOR_MODLFOTOFILTERFC) != 0
449                         || getGeneratorValue(generators,
450                             SF2Region.GENERATOR_MODLFOTOPITCH) != 0
451                         || getGeneratorValue(generators,
452                             SF2Region.GENERATOR_MODLFOTOVOLUME) != 0) {
453                     short lfo_freq = getGeneratorValue(generators,
454                             SF2Region.GENERATOR_FREQMODLFO);
455                     short lfo_delay = getGeneratorValue(generators,
456                             SF2Region.GENERATOR_DELAYMODLFO);
457                     addTimecentValue(performer,
458                             ModelDestination.DESTINATION_LFO1_DELAY, lfo_delay);
459                     addValue(performer,
460                             ModelDestination.DESTINATION_LFO1_FREQ, lfo_freq);
461                 }
462
463                 short vib_freq = getGeneratorValue(generators,
464                         SF2Region.GENERATOR_FREQVIBLFO);
465                 short vib_delay = getGeneratorValue(generators,
466                         SF2Region.GENERATOR_DELAYVIBLFO);
467                 addTimecentValue(performer,
468                         ModelDestination.DESTINATION_LFO2_DELAY, vib_delay);
469                 addValue(performer,
470                         ModelDestination.DESTINATION_LFO2_FREQ, vib_freq);
471
472
473                 if (getGeneratorValue(generators,
474                         SF2Region.GENERATOR_VIBLFOTOPITCH) != 0) {
475                     double fvalue = getGeneratorValue(generators,
476                             SF2Region.GENERATOR_VIBLFOTOPITCH);
477                     ModelIdentifier src = ModelSource.SOURCE_LFO2;
478                     ModelIdentifier dest = ModelDestination.DESTINATION_PITCH;
479                     performer.getConnectionBlocks().add(
480                         new ModelConnectionBlock(
481                             new ModelSource(src,
482                                 ModelStandardTransform.DIRECTION_MIN2MAX,
483                                 ModelStandardTransform.POLARITY_BIPOLAR),
484                             fvalue, new ModelDestination(dest)));
485                 }
486
487                 if (getGeneratorValue(generators,
488                         SF2Region.GENERATOR_MODLFOTOFILTERFC) != 0) {
489                     double fvalue = getGeneratorValue(generators,
490                             SF2Region.GENERATOR_MODLFOTOFILTERFC);
491                     ModelIdentifier src = ModelSource.SOURCE_LFO1;
492                     ModelIdentifier dest = ModelDestination.DESTINATION_FILTER_FREQ;
493                     performer.getConnectionBlocks().add(
494                         new ModelConnectionBlock(
```

```
                    new ModelSource(src,
                        ModelStandardTransform.DIRECTION_MIN2MAX,
                        ModelStandardTransform.POLARITY_BIPOLAR),
                    fvalue, new ModelDestination(dest)));
            }

            if (getGeneratorValue(generators,
                    SF2Region.GENERATOR_MODLFOTOPITCH) != 0) {
                double fvalue = getGeneratorValue(generators,
                        SF2Region.GENERATOR_MODLFOTOPITCH);
                ModelIdentifier src = ModelSource.SOURCE_LFO1;
                ModelIdentifier dest = ModelDestination.DESTINATION_PITCH;
                performer.getConnectionBlocks().add(
                    new ModelConnectionBlock(
                        new ModelSource(src,
                            ModelStandardTransform.DIRECTION_MIN2MAX,
                            ModelStandardTransform.POLARITY_BIPOLAR),
                        fvalue, new ModelDestination(dest)));
            }

            if (getGeneratorValue(generators,
                    SF2Region.GENERATOR_MODLFOTOVOLUME) != 0) {
                double fvalue = getGeneratorValue(generators,
                        SF2Region.GENERATOR_MODLFOTOVOLUME);
                ModelIdentifier src = ModelSource.SOURCE_LFO1;
                ModelIdentifier dest = ModelDestination.DESTINATION_GAIN;
                performer.getConnectionBlocks().add(
                    new ModelConnectionBlock(
                        new ModelSource(src,
                            ModelStandardTransform.DIRECTION_MIN2MAX,
                            ModelStandardTransform.POLARITY_BIPOLAR),
                        fvalue, new ModelDestination(dest)));
            }

            if (layerzone.getShort(SF2Region.GENERATOR_KEYNUM) != -1) {
                double val = layerzone.getShort(SF2Region.GENERATOR_KEYNUM)/128.0;
                addValue(performer, ModelDestination.DESTINATION_KEYNUMBER, val);
            }

            if (layerzone.getShort(SF2Region.GENERATOR_VELOCITY) != -1) {
                double val = layerzone.getShort(SF2Region.GENERATOR_VELOCITY)
                             / 128.0;
                addValue(performer, ModelDestination.DESTINATION_VELOCITY, val);
            }

            if (getGeneratorValue(generators,
                    SF2Region.GENERATOR_INITIALFILTERFC) < 13500) {
                short filter_freq = getGeneratorValue(generators,
                        SF2Region.GENERATOR_INITIALFILTERFC);
                short filter_q = getGeneratorValue(generators,
                        SF2Region.GENERATOR_INITIALFILTERQ);
                addValue(performer,
                        ModelDestination.DESTINATION_FILTER_FREQ, filter_freq);
                addValue(performer,
                        ModelDestination.DESTINATION_FILTER_Q, filter_q);
            }

            int tune = 100 * getGeneratorValue(generators,
                    SF2Region.GENERATOR_COARSETUNE);
            tune += getGeneratorValue(generators,
                    SF2Region.GENERATOR_FINETUNE);
            if (tune != 0) {
```

```
                addValue(performer,
                        ModelDestination.DESTINATION_PITCH, (short) tune);
            }
            if (getGeneratorValue(generators, SF2Region.GENERATOR_PAN) != 0) {
                short val = getGeneratorValue(generators,
                        SF2Region.GENERATOR_PAN);
                addValue(performer, ModelDestination.DESTINATION_PAN, val);
            }
            if (getGeneratorValue(generators, SF2Region.GENERATOR_INITIALATTENUATION) != 0) {
                short val = getGeneratorValue(generators,
                        SF2Region.GENERATOR_INITIALATTENUATION);
                addValue(performer,
                        ModelDestination.DESTINATION_GAIN, -0.376287f * val);
            }
            if (getGeneratorValue(generators,
                    SF2Region.GENERATOR_CHORUSEFFECTSSEND) != 0) {
                short val = getGeneratorValue(generators,
                        SF2Region.GENERATOR_CHORUSEFFECTSSEND);
                addValue(performer, ModelDestination.DESTINATION_CHORUS, val);
            }
            if (getGeneratorValue(generators,
                    SF2Region.GENERATOR_REVERBEFFECTSSEND) != 0) {
                short val = getGeneratorValue(generators,
                        SF2Region.GENERATOR_REVERBEFFECTSSEND);
                addValue(performer, ModelDestination.DESTINATION_REVERB, val);
            }
            if (getGeneratorValue(generators,
                    SF2Region.GENERATOR_SCALETUNING) != 100) {
                short fvalue = getGeneratorValue(generators,
                        SF2Region.GENERATOR_SCALETUNING);
                if (fvalue == 0) {
                    ModelIdentifier dest = ModelDestination.DESTINATION_PITCH;
                    performer.getConnectionBlocks().add(
                        new ModelConnectionBlock(null, rootkey * 100,
                            new ModelDestination(dest)));
                } else {
                    ModelIdentifier dest = ModelDestination.DESTINATION_PITCH;
                    performer.getConnectionBlocks().add(
                        new ModelConnectionBlock(null, rootkey * (100 - fvalue),
                            new ModelDestination(dest)));
                }

                ModelIdentifier src = ModelSource.SOURCE_NOTEON_KEYNUMBER;
                ModelIdentifier dest = ModelDestination.DESTINATION_PITCH;
                performer.getConnectionBlocks().add(
                    new ModelConnectionBlock(new ModelSource(src),
                        128 * fvalue, new ModelDestination(dest)));

            }

            performer.getConnectionBlocks().add(
                new ModelConnectionBlock(
                    new ModelSource(ModelSource.SOURCE_NOTEON_VELOCITY,
                        new ModelTransform() {
                            public double transform(double value) {
                                if (value < 0.5)
                                    return 1 - value * 2;
                                else
                                    return 0;
                            }
                        }),
                    -2400,
```

```
                    new ModelDestination(
                        ModelDestination.DESTINATION_FILTER_FREQ)));


            performer.getConnectionBlocks().add(
                new ModelConnectionBlock(
                    new ModelSource(ModelSource.SOURCE_LFO2,
                        ModelStandardTransform.DIRECTION_MIN2MAX,
                        ModelStandardTransform.POLARITY_BIPOLAR,
                        ModelStandardTransform.TRANSFORM_LINEAR),
                    new ModelSource(new ModelIdentifier("midi_cc", "1", 0),
                        ModelStandardTransform.DIRECTION_MIN2MAX,
                        ModelStandardTransform.POLARITY_UNIPOLAR,
                        ModelStandardTransform.TRANSFORM_LINEAR),
                    50, new ModelDestination(
                        ModelDestination.DESTINATION_PITCH)));

            if (layer.getGlobalRegion() != null) {
                for (SF2Modulator modulator
                        : layer.getGlobalRegion().getModulators()) {
                    convertModulator(performer, modulator);
                }
            }
            for (SF2Modulator modulator : layerzone.getModulators())
                convertModulator(performer, modulator);

            if (presetglobal != null) {
                for (SF2Modulator modulator : presetglobal.getModulators())
                    convertModulator(performer, modulator);
            }
            for (SF2Modulator modulator : presetzone.getModulators())
                convertModulator(performer, modulator);

        }
    }
    return performers;
}

private void convertModulator(ModelPerformer performer,
        SF2Modulator modulator) {
    ModelSource src1 = convertSource(modulator.getSourceOperator());
    ModelSource src2 = convertSource(modulator.getAmountSourceOperator());
    if (src1 == null && modulator.getSourceOperator() != 0)
        return;
    if (src2 == null && modulator.getAmountSourceOperator() != 0)
        return;
    double amount = modulator.getAmount();
    double[] amountcorrection = new double[1];
    ModelSource[] extrasrc = new ModelSource[1];
    amountcorrection[0] = 1;
    ModelDestination dst = convertDestination(
            modulator.getDestinationOperator(), amountcorrection, extrasrc);
    amount *= amountcorrection[0];
    if (dst == null)
        return;
    if (modulator.getTransportOperator() == SF2Modulator.TRANSFORM_ABSOLUTE) {
        ((ModelStandardTransform)dst.getTransform()).setTransform(
                ModelStandardTransform.TRANSFORM_ABSOLUTE);
    }
    ModelConnectionBlock conn = new ModelConnectionBlock(src1, src2, amount, dst);
    if (extrasrc[0] != null)
        conn.addSource(extrasrc[0]);
```

```
681         performer.getConnectionBlocks().add(conn);
682
683     }
684
685     private static ModelSource convertSource(int src) {
686         if (src == 0)
687             return null;
688         ModelIdentifier id = null;
689         int idsrc = src & 0x7F;
690         if ((src & SF2Modulator.SOURCE_MIDI_CONTROL) != 0) {
691             id = new ModelIdentifier("midi_cc", Integer.toString(idsrc));
692         } else {
693             if (idsrc == SF2Modulator.SOURCE_NOTE_ON_VELOCITY)
694                 id = ModelSource.SOURCE_NOTEON_VELOCITY;
695             if (idsrc == SF2Modulator.SOURCE_NOTE_ON_KEYNUMBER)
696                 id = ModelSource.SOURCE_NOTEON_KEYNUMBER;
697             if (idsrc == SF2Modulator.SOURCE_POLY_PRESSURE)
698                 id = ModelSource.SOURCE_MIDI_POLY_PRESSURE;
699             if (idsrc == SF2Modulator.SOURCE_CHANNEL_PRESSURE)
700                 id = ModelSource.SOURCE_MIDI_CHANNEL_PRESSURE;
701             if (idsrc == SF2Modulator.SOURCE_PITCH_WHEEL)
702                 id = ModelSource.SOURCE_MIDI_PITCH;
703             if (idsrc == SF2Modulator.SOURCE_PITCH_SENSITIVITY)
704                 id = new ModelIdentifier("midi_rpn", "0");
705         }
706         if (id == null)
707             return null;
708
709         ModelSource msrc = new ModelSource(id);
710         ModelStandardTransform transform
711                 = (ModelStandardTransform) msrc.getTransform();
712
713         if ((SF2Modulator.SOURCE_DIRECTION_MAX_MIN & src) != 0)
714             transform.setDirection(ModelStandardTransform.DIRECTION_MAX2MIN);
715         else
716             transform.setDirection(ModelStandardTransform.DIRECTION_MIN2MAX);
717
718         if ((SF2Modulator.SOURCE_POLARITY_BIPOLAR & src) != 0)
719             transform.setPolarity(ModelStandardTransform.POLARITY_BIPOLAR);
720         else
721             transform.setPolarity(ModelStandardTransform.POLARITY_UNIPOLAR);
722
723         if ((SF2Modulator.SOURCE_TYPE_CONCAVE & src) != 0)
724             transform.setTransform(ModelStandardTransform.TRANSFORM_CONCAVE);
725         if ((SF2Modulator.SOURCE_TYPE_CONVEX & src) != 0)
726             transform.setTransform(ModelStandardTransform.TRANSFORM_CONVEX);
727         if ((SF2Modulator.SOURCE_TYPE_SWITCH & src) != 0)
728             transform.setTransform(ModelStandardTransform.TRANSFORM_SWITCH);
729
730         return msrc;
731     }
732
733     protected static ModelDestination convertDestination(int dst,
734             double[] amountcorrection, ModelSource[] extrasrc) {
735         ModelIdentifier id = null;
736         switch (dst) {
737             case SF2Region.GENERATOR_INITIALFILTERFC:
738                 id = ModelDestination.DESTINATION_FILTER_FREQ;
739                 break;
740             case SF2Region.GENERATOR_INITIALFILTERQ:
741                 id = ModelDestination.DESTINATION_FILTER_Q;
742                 break;
```

```
            case SF2Region.GENERATOR_CHORUSEFFECTSSEND:
                id = ModelDestination.DESTINATION_CHORUS;
                break;
            case SF2Region.GENERATOR_REVERBEFFECTSSEND:
                id = ModelDestination.DESTINATION_REVERB;
                break;
            case SF2Region.GENERATOR_PAN:
                id = ModelDestination.DESTINATION_PAN;
                break;
            case SF2Region.GENERATOR_DELAYMODLFO:
                id = ModelDestination.DESTINATION_LFO1_DELAY;
                break;
            case SF2Region.GENERATOR_FREQMODLFO:
                id = ModelDestination.DESTINATION_LFO1_FREQ;
                break;
            case SF2Region.GENERATOR_DELAYVIBLFO:
                id = ModelDestination.DESTINATION_LFO2_DELAY;
                break;
            case SF2Region.GENERATOR_FREQVIBLFO:
                id = ModelDestination.DESTINATION_LFO2_FREQ;
                break;

            case SF2Region.GENERATOR_DELAYMODENV:
                id = ModelDestination.DESTINATION_EG2_DELAY;
                break;
            case SF2Region.GENERATOR_ATTACKMODENV:
                id = ModelDestination.DESTINATION_EG2_ATTACK;
                break;
            case SF2Region.GENERATOR_HOLDMODENV:
                id = ModelDestination.DESTINATION_EG2_HOLD;
                break;
            case SF2Region.GENERATOR_DECAYMODENV:
                id = ModelDestination.DESTINATION_EG2_DECAY;
                break;
            case SF2Region.GENERATOR_SUSTAINMODENV:
                id = ModelDestination.DESTINATION_EG2_SUSTAIN;
                amountcorrection[0] = -1;
                break;
            case SF2Region.GENERATOR_RELEASEMODENV:
                id = ModelDestination.DESTINATION_EG2_RELEASE;
                break;
            case SF2Region.GENERATOR_DELAYVOLENV:
                id = ModelDestination.DESTINATION_EG1_DELAY;
                break;
            case SF2Region.GENERATOR_ATTACKVOLENV:
                id = ModelDestination.DESTINATION_EG1_ATTACK;
                break;
            case SF2Region.GENERATOR_HOLDVOLENV:
                id = ModelDestination.DESTINATION_EG1_HOLD;
                break;
            case SF2Region.GENERATOR_DECAYVOLENV:
                id = ModelDestination.DESTINATION_EG1_DECAY;
                break;
            case SF2Region.GENERATOR_SUSTAINVOLENV:
                id = ModelDestination.DESTINATION_EG1_SUSTAIN;
                amountcorrection[0] = -1;
                break;
            case SF2Region.GENERATOR_RELEASEVOLENV:
                id = ModelDestination.DESTINATION_EG1_RELEASE;
                break;
            case SF2Region.GENERATOR_KEYNUM:
                id = ModelDestination.DESTINATION_KEYNUMBER;
```

```
805                 break;
806             case SF2Region.GENERATOR_VELOCITY:
807                 id = ModelDestination.DESTINATION_VELOCITY;
808                 break;
809
810             case SF2Region.GENERATOR_COARSETUNE:
811                 amountcorrection[0] = 100;
812                 id = ModelDestination.DESTINATION_PITCH;
813                 break;
814
815             case SF2Region.GENERATOR_FINETUNE:
816                 id = ModelDestination.DESTINATION_PITCH;
817                 break;
818
819             case SF2Region.GENERATOR_INITIALATTENUATION:
820                 id = ModelDestination.DESTINATION_GAIN;
821                 amountcorrection[0] = -0.376287f;
822                 break;
823
824             case SF2Region.GENERATOR_VIBLFOTOPITCH:
825                 id = ModelDestination.DESTINATION_PITCH;
826                 extrasrc[0] = new ModelSource(
827                         ModelSource.SOURCE_LFO2,
828                         ModelStandardTransform.DIRECTION_MIN2MAX,
829                         ModelStandardTransform.POLARITY_BIPOLAR);
830                 break;
831
832             case SF2Region.GENERATOR_MODLFOTOPITCH:
833                 id = ModelDestination.DESTINATION_PITCH;
834                 extrasrc[0] = new ModelSource(
835                         ModelSource.SOURCE_LFO1,
836                         ModelStandardTransform.DIRECTION_MIN2MAX,
837                         ModelStandardTransform.POLARITY_BIPOLAR);
838                 break;
839
840             case SF2Region.GENERATOR_MODLFOTOFILTERFC:
841                 id = ModelDestination.DESTINATION_FILTER_FREQ;
842                 extrasrc[0] = new ModelSource(
843                         ModelSource.SOURCE_LFO1,
844                         ModelStandardTransform.DIRECTION_MIN2MAX,
845                         ModelStandardTransform.POLARITY_BIPOLAR);
846                 break;
847
848             case SF2Region.GENERATOR_MODLFOTOVOLUME:
849                 id = ModelDestination.DESTINATION_GAIN;
850                 amountcorrection[0] = -0.376287f;
851                 extrasrc[0] = new ModelSource(
852                         ModelSource.SOURCE_LFO1,
853                         ModelStandardTransform.DIRECTION_MIN2MAX,
854                         ModelStandardTransform.POLARITY_BIPOLAR);
855                 break;
856
857             case SF2Region.GENERATOR_MODENVTOPITCH:
858                 id = ModelDestination.DESTINATION_PITCH;
859                 extrasrc[0] = new ModelSource(
860                         ModelSource.SOURCE_EG2,
861                         ModelStandardTransform.DIRECTION_MIN2MAX,
862                         ModelStandardTransform.POLARITY_BIPOLAR);
863                 break;
864
865             case SF2Region.GENERATOR_MODENVTOFILTERFC:
866                 id = ModelDestination.DESTINATION_FILTER_FREQ;
```

```
                extrasrc[0] = new ModelSource(
                        ModelSource.SOURCE_EG2,
                        ModelStandardTransform.DIRECTION_MIN2MAX,
                        ModelStandardTransform.POLARITY_BIPOLAR);
                break;

            default:
                break;
        }
        if (id != null)
            return new ModelDestination(id);
        return null;
    }

    private void addTimecentValue(ModelPerformer performer,
            ModelIdentifier dest, short value) {
        double fvalue;
        if (value == -12000)
            fvalue = Double.NEGATIVE_INFINITY;
        else
            fvalue = value;
        performer.getConnectionBlocks().add(
                new ModelConnectionBlock(fvalue, new ModelDestination(dest)));
    }

    private void addValue(ModelPerformer performer,
            ModelIdentifier dest, short value) {
        double fvalue = value;
        performer.getConnectionBlocks().add(
                new ModelConnectionBlock(fvalue, new ModelDestination(dest)));
    }

    private void addValue(ModelPerformer performer,
            ModelIdentifier dest, double value) {
        double fvalue = value;
        performer.getConnectionBlocks().add(
                new ModelConnectionBlock(fvalue, new ModelDestination(dest)));
    }

    private short getGeneratorValue(Map<Integer, Short> generators, int gen) {
        if (generators.containsKey(gen))
            return generators.get(gen);
        return SF2Region.getDefaultValue(gen);
    }
}
```

67 com/sun/media/sound/SF2InstrumentRegion.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * Soundfont instrument region.
 *
 * @author Karl Helgason
 */
public class SF2InstrumentRegion extends SF2Region {

    protected SF2Layer layer;

    public SF2Layer getLayer() {
        return layer;
    }

    public void setLayer(SF2Layer layer) {
        this.layer = layer;
    }
}
```

68 com/sun/media/sound/SF2Layer.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

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

import javax.sound.midi.SoundbankResource;

/**
 * Soundfont layer.
 *
 * @author Karl Helgason
 */
public class SF2Layer extends SoundbankResource {

    protected String name = "";
    protected SF2GlobalRegion globalregion = null;
    protected List<SF2LayerRegion> regions = new ArrayList<SF2LayerRegion>();

    public SF2Layer(SF2Soundbank soundBank) {
        super(soundBank, null, null);
    }

    public SF2Layer() {
        super(null, null, null);
    }

    public Object getData() {
        return null;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
```

```
    }

    public List<SF2LayerRegion> getRegions() {
        return regions;
    }

    public SF2GlobalRegion getGlobalRegion() {
        return globalregion;
    }

    public void setGlobalZone(SF2GlobalRegion zone) {
        globalregion = zone;
    }

    public String toString() {
        return "Layer: " + name;
    }
}
```

69 com/sun/media/sound/SF2LayerRegion.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * Soundfont layer region.
 *
 * @author Karl Helgason
 */
public class SF2LayerRegion extends SF2Region {

    protected SF2Sample sample;

    public SF2Sample getSample() {
        return sample;
    }

    public void setSample(SF2Sample sample) {
        this.sample = sample;
    }
}
```

70 com/sun/media/sound/SF2Modulator.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * Soundfont modulator container.
 *
 * @author Karl Helgason
 */
public class SF2Modulator {

    public final static int SOURCE_NONE = 0;
    public final static int SOURCE_NOTE_ON_VELOCITY = 2;
    public final static int SOURCE_NOTE_ON_KEYNUMBER = 3;
    public final static int SOURCE_POLY_PRESSURE = 10;
    public final static int SOURCE_CHANNEL_PRESSURE = 13;
    public final static int SOURCE_PITCH_WHEEL = 14;
    public final static int SOURCE_PITCH_SENSITIVITY = 16;
    public final static int SOURCE_MIDI_CONTROL = 128 * 1;
    public final static int SOURCE_DIRECTION_MIN_MAX = 256 * 0;
    public final static int SOURCE_DIRECTION_MAX_MIN = 256 * 1;
    public final static int SOURCE_POLARITY_UNIPOLAR = 512 * 0;
    public final static int SOURCE_POLARITY_BIPOLAR = 512 * 1;
    public final static int SOURCE_TYPE_LINEAR = 1024 * 0;
    public final static int SOURCE_TYPE_CONCAVE = 1024 * 1;
    public final static int SOURCE_TYPE_CONVEX = 1024 * 2;
    public final static int SOURCE_TYPE_SWITCH = 1024 * 3;
    public final static int TRANSFORM_LINEAR = 0;
    public final static int TRANSFORM_ABSOLUTE = 2;
    protected int sourceOperator;
    protected int destinationOperator;
    protected short amount;
    protected int amountSourceOperator;
    protected int transportOperator;

    public short getAmount() {
        return amount;
    }
```

```
61
62      public void setAmount(short amount) {
63          this.amount = amount;
64      }
65
66      public int getAmountSourceOperator() {
67          return amountSourceOperator;
68      }
69
70      public void setAmountSourceOperator(int amountSourceOperator) {
71          this.amountSourceOperator = amountSourceOperator;
72      }
73
74      public int getTransportOperator() {
75          return transportOperator;
76      }
77
78      public void setTransportOperator(int transportOperator) {
79          this.transportOperator = transportOperator;
80      }
81
82      public int getDestinationOperator() {
83          return destinationOperator;
84      }
85
86      public void setDestinationOperator(int destinationOperator) {
87          this.destinationOperator = destinationOperator;
88      }
89
90      public int getSourceOperator() {
91          return sourceOperator;
92      }
93
94      public void setSourceOperator(int sourceOperator) {
95          this.sourceOperator = sourceOperator;
96      }
97  }
```

71 com/sun/media/sound/SF2Region.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * Soundfont general region.
 *
 * @author Karl Helgason
 */
public class SF2Region {

    public final static int GENERATOR_STARTADDRSOFFSET = 0;
    public final static int GENERATOR_ENDADDRSOFFSET = 1;
    public final static int GENERATOR_STARTLOOPADDRSOFFSET = 2;
    public final static int GENERATOR_ENDLOOPADDRSOFFSET = 3;
    public final static int GENERATOR_STARTADDRSCOARSEOFFSET = 4;
    public final static int GENERATOR_MODLFOTOPITCH = 5;
    public final static int GENERATOR_VIBLFOTOPITCH = 6;
    public final static int GENERATOR_MODENVTOPITCH = 7;
    public final static int GENERATOR_INITIALFILTERFC = 8;
    public final static int GENERATOR_INITIALFILTERQ = 9;
    public final static int GENERATOR_MODLFOTOFILTERFC = 10;
    public final static int GENERATOR_MODENVTOFILTERFC = 11;
    public final static int GENERATOR_ENDADDRSCOARSEOFFSET = 12;
    public final static int GENERATOR_MODLFOTOVOLUME = 13;
    public final static int GENERATOR_UNUSED1 = 14;
    public final static int GENERATOR_CHORUSEFFECTSSEND = 15;
    public final static int GENERATOR_REVERBEFFECTSSEND = 16;
    public final static int GENERATOR_PAN = 17;
    public final static int GENERATOR_UNUSED2 = 18;
    public final static int GENERATOR_UNUSED3 = 19;
    public final static int GENERATOR_UNUSED4 = 20;
    public final static int GENERATOR_DELAYMODLFO = 21;
```

```
    public final static int GENERATOR_FREQMODLFO = 22;
    public final static int GENERATOR_DELAYVIBLFO = 23;
    public final static int GENERATOR_FREQVIBLFO = 24;
    public final static int GENERATOR_DELAYMODENV = 25;
    public final static int GENERATOR_ATTACKMODENV = 26;
    public final static int GENERATOR_HOLDMODENV = 27;
    public final static int GENERATOR_DECAYMODENV = 28;
    public final static int GENERATOR_SUSTAINMODENV = 29;
    public final static int GENERATOR_RELEASEMODENV = 30;
    public final static int GENERATOR_KEYNUMTOMODENVHOLD = 31;
    public final static int GENERATOR_KEYNUMTOMODENVDECAY = 32;
    public final static int GENERATOR_DELAYVOLENV = 33;
    public final static int GENERATOR_ATTACKVOLENV = 34;
    public final static int GENERATOR_HOLDVOLENV = 35;
    public final static int GENERATOR_DECAYVOLENV = 36;
    public final static int GENERATOR_SUSTAINVOLENV = 37;
    public final static int GENERATOR_RELEASEVOLENV = 38;
    public final static int GENERATOR_KEYNUMTOVOLENVHOLD = 39;
    public final static int GENERATOR_KEYNUMTOVOLENVDECAY = 40;
    public final static int GENERATOR_INSTRUMENT = 41;
    public final static int GENERATOR_RESERVED1 = 42;
    public final static int GENERATOR_KEYRANGE = 43;
    public final static int GENERATOR_VELRANGE = 44;
    public final static int GENERATOR_STARTLOOPADDRSCOARSEOFFSET = 45;
    public final static int GENERATOR_KEYNUM = 46;
    public final static int GENERATOR_VELOCITY = 47;
    public final static int GENERATOR_INITIALATTENUATION = 48;
    public final static int GENERATOR_RESERVED2 = 49;
    public final static int GENERATOR_ENDLOOPADDRSCOARSEOFFSET = 50;
    public final static int GENERATOR_COARSETUNE = 51;
    public final static int GENERATOR_FINETUNE = 52;
    public final static int GENERATOR_SAMPLEID = 53;
    public final static int GENERATOR_SAMPLEMODES = 54;
    public final static int GENERATOR_RESERVED3 = 55;
    public final static int GENERATOR_SCALETUNING = 56;
    public final static int GENERATOR_EXCLUSIVECLASS = 57;
    public final static int GENERATOR_OVERRIDINGROOTKEY = 58;
    public final static int GENERATOR_UNUSED5 = 59;
    public final static int GENERATOR_ENDOPR = 60;
    protected Map<Integer, Short> generators = new HashMap<Integer, Short>();
    protected List<SF2Modulator> modulators = new ArrayList<SF2Modulator>();

    public Map<Integer, Short> getGenerators() {
        return generators;
    }

    public boolean contains(int generator) {
        return generators.containsKey(generator);
    }

    static public short getDefaultValue(int generator) {
        if (generator == 8) return (short)13500;
        if (generator == 21) return (short)-12000;
        if (generator == 23) return (short)-12000;
        if (generator == 25) return (short)-12000;
        if (generator == 26) return (short)-12000;
        if (generator == 27) return (short)-12000;
        if (generator == 28) return (short)-12000;
        if (generator == 30) return (short)-12000;
        if (generator == 33) return (short)-12000;
        if (generator == 34) return (short)-12000;
        if (generator == 35) return (short)-12000;
```

```
123         if (generator == 36) return (short)-12000;
124         if (generator == 38) return (short)-12000;
125         if (generator == 43) return (short)0x7F00;
126         if (generator == 44) return (short)0x7F00;
127         if (generator == 46) return (short)-1;
128         if (generator == 47) return (short)-1;
129         if (generator == 56) return (short)100;
130         if (generator == 58) return (short)-1;
131         return 0;
132     }
133 
134     public short getShort(int generator) {
135         if (!contains(generator))
136             return getDefaultValue(generator);
137         return generators.get(generator);
138     }
139 
140     public void putShort(int generator, short value) {
141         generators.put(generator, value);
142     }
143 
144     public byte[] getBytes(int generator) {
145         int val = getInteger(generator);
146         byte[] bytes = new byte[2];
147         bytes[0] = (byte) (0xFF & val);
148         bytes[1] = (byte) ((0xFF00 & val) >> 8);
149         return bytes;
150     }
151 
152     public void putBytes(int generator, byte[] bytes) {
153         generators.put(generator, (short) (bytes[0] + (bytes[1] << 8)));
154     }
155 
156     public int getInteger(int generator) {
157         return 0xFFFF & getShort(generator);
158     }
159 
160     public void putInteger(int generator, int value) {
161         generators.put(generator, (short) value);
162     }
163 
164     public List<SF2Modulator> getModulators() {
165         return modulators;
166     }
167 }
```

72 com/sun/media/sound/SF2Sample.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.io.InputStream;

import javax.sound.midi.Soundbank;
import javax.sound.midi.SoundbankResource;
import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;

/**
 * Soundfont sample storage.
 *
 * @author Karl Helgason
 */
public class SF2Sample extends SoundbankResource {

    protected String name = "";
    protected long startLoop = 0;
    protected long endLoop = 0;
    protected long sampleRate = 44100;
    protected int originalPitch = 60;
    protected byte pitchCorrection = 0;
    protected int sampleLink = 0;
    protected int sampleType = 0;
    protected ModelByteBuffer data;
    protected ModelByteBuffer data24;

    public SF2Sample(Soundbank soundBank) {
        super(soundBank, null, AudioInputStream.class);
    }

    public SF2Sample() {
        super(null, null, AudioInputStream.class);
    }

    public Object getData() {
```

```
        AudioFormat format = getFormat();
        /*
        if (sampleFile != null) {
            FileInputStream fis;
            try {
                fis = new FileInputStream(sampleFile);
                RIFFReader riff = new RIFFReader(fis);
                if (!riff.getFormat().equals("RIFF")) {
                    throw new RIFFInvalidDataException(
                        "Input stream is not a valid RIFF stream!");
                }
                if (!riff.getType().equals("sfbk")) {
                    throw new RIFFInvalidDataException(
                        "Input stream is not a valid SoundFont!");
                }
                while (riff.hasNextChunk()) {
                    RIFFReader chunk = riff.nextChunk();
                    if (chunk.getFormat().equals("LIST")) {
                        if (chunk.getType().equals("sdta")) {
                            while(chunk.hasNextChunk()) {
                                RIFFReader chunkchunk = chunk.nextChunk();
                                if(chunkchunk.getFormat().equals("smpl")) {
                                    chunkchunk.skip(sampleOffset);
                                    return new AudioInputStream(chunkchunk,
                                            format, sampleLen);
                                }
                            }
                        }
                    }
                }
                return null;
            } catch (Exception e) {
                return new Throwable(e.toString());
            }
        }
        */
        InputStream is = data.getInputStream();
        if (is == null)
            return null;
        return new AudioInputStream(is, format, data.capacity());
    }

    public ModelByteBuffer getDataBuffer() {
        return data;
    }

    public ModelByteBuffer getData24Buffer() {
        return data24;
    }

    public AudioFormat getFormat() {
        return new AudioFormat(sampleRate, 16, 1, true, false);
    }

    public void setData(ModelByteBuffer data) {
        this.data = data;
    }

    public void setData(byte[] data) {
        this.data = new ModelByteBuffer(data);
    }
```

```
123
124     public void setData(byte[] data, int offset, int length) {
125         this.data = new ModelByteBuffer(data, offset, length);
126     }
127
128     public void setData24(ModelByteBuffer data24) {
129         this.data24 = data24;
130     }
131
132     public void setData24(byte[] data24) {
133         this.data24 = new ModelByteBuffer(data24);
134     }
135
136     public void setData24(byte[] data24, int offset, int length) {
137         this.data24 = new ModelByteBuffer(data24, offset, length);
138     }
139
140     /*
141     public void setData(File file, int offset, int length) {
142         this.data = null;
143         this.sampleFile = file;
144         this.sampleOffset = offset;
145         this.sampleLen = length;
146     }
147     */
148
149     public String getName() {
150         return name;
151     }
152
153     public void setName(String name) {
154         this.name = name;
155     }
156
157     public long getEndLoop() {
158         return endLoop;
159     }
160
161     public void setEndLoop(long endLoop) {
162         this.endLoop = endLoop;
163     }
164
165     public int getOriginalPitch() {
166         return originalPitch;
167     }
168
169     public void setOriginalPitch(int originalPitch) {
170         this.originalPitch = originalPitch;
171     }
172
173     public byte getPitchCorrection() {
174         return pitchCorrection;
175     }
176
177     public void setPitchCorrection(byte pitchCorrection) {
178         this.pitchCorrection = pitchCorrection;
179     }
180
181     public int getSampleLink() {
182         return sampleLink;
183     }
184
```

```
    public void setSampleLink(int sampleLink) {
        this.sampleLink = sampleLink;
    }

    public long getSampleRate() {
        return sampleRate;
    }

    public void setSampleRate(long sampleRate) {
        this.sampleRate = sampleRate;
    }

    public int getSampleType() {
        return sampleType;
    }

    public void setSampleType(int sampleType) {
        this.sampleType = sampleType;
    }

    public long getStartLoop() {
        return startLoop;
    }

    public void setStartLoop(long startLoop) {
        this.startLoop = startLoop;
    }

    public String toString() {
        return "Sample: " + name;
    }
}
```

73 com/sun/media/sound/SF2Soundbank.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.URL;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

import javax.sound.midi.Instrument;
import javax.sound.midi.Patch;
import javax.sound.midi.Soundbank;
import javax.sound.midi.SoundbankResource;

/**
 * A SoundFont 2.04 soundbank reader.
 *
 * Based on SoundFont 2.04 specification from:
 * <p>  http://developer.creative.com <br>
 *      http://www.soundfont.com/ ;
 *
 * @author Karl Helgason
 */
public class SF2Soundbank implements Soundbank {

    // version of the Sound Font RIFF file
    protected int major = 2;
    protected int minor = 1;
    // target Sound Engine
    protected String targetEngine = "EMU8000";
    // Sound Font Bank Name
```

```
    protected String name = "untitled";
    // Sound ROM Name
    protected String romName = null;
    // Sound ROM Version
    protected int romVersionMajor = -1;
    protected int romVersionMinor = -1;
    // Date of Creation of the Bank
    protected String creationDate = null;
    // Sound Designers and Engineers for the Bank
    protected String engineers = null;
    // Product for which the Bank was intended
    protected String product = null;
    // Copyright message
    protected String copyright = null;
    // Comments
    protected String comments = null;
    // The SoundFont tools used to create and alter the bank
    protected String tools = null;
    // The Sample Data loaded from the SoundFont
    private ModelByteBuffer sampleData = null;
    private ModelByteBuffer sampleData24 = null;
    private File sampleFile = null;
    private boolean largeFormat = false;
    private List<SF2Instrument> instruments = new ArrayList<SF2Instrument>();
    private List<SF2Layer> layers = new ArrayList<SF2Layer>();
    private List<SF2Sample> samples = new ArrayList<SF2Sample>();

    public SF2Soundbank() {
    }

    public SF2Soundbank(URL url) throws IOException {

        InputStream is = url.openStream();
        try {
            readSoundbank(is);
        } finally {
            is.close();
        }
    }

    public SF2Soundbank(File file) throws IOException {
        largeFormat = true;
        sampleFile = file;
        InputStream is = new FileInputStream(file);
        try {
            readSoundbank(is);
        } finally {
            is.close();
        }
    }

    public SF2Soundbank(InputStream inputstream) throws IOException {
        readSoundbank(inputstream);
    }

    private void readSoundbank(InputStream inputstream) throws IOException {
        RIFFReader riff = new RIFFReader(inputstream);
        if (!riff.getFormat().equals("RIFF")) {
            throw new RIFFInvalidFormatException(
                    "Input␣stream␣is␣not␣a␣valid␣RIFF␣stream!");
        }
        if (!riff.getType().equals("sfbk")) {
```

```
123             throw new RIFFInvalidFormatException(
124                     "Input stream is not a valid SoundFont!");
125         }
126         while (riff.hasNextChunk()) {
127             RIFFReader chunk = riff.nextChunk();
128             if (chunk.getFormat().equals("LIST")) {
129                 if (chunk.getType().equals("INFO"))
130                     readInfoChunk(chunk);
131                 if (chunk.getType().equals("sdta"))
132                     readSdtaChunk(chunk);
133                 if (chunk.getType().equals("pdta"))
134                     readPdtaChunk(chunk);
135             }
136         }
137     }
138 
139     private void readInfoChunk(RIFFReader riff) throws IOException {
140         while (riff.hasNextChunk()) {
141             RIFFReader chunk = riff.nextChunk();
142             String format = chunk.getFormat();
143             if (format.equals("ifil")) {
144                 major = chunk.readUnsignedShort();
145                 minor = chunk.readUnsignedShort();
146             } else if (format.equals("isng")) {
147                 this.targetEngine = chunk.readString(chunk.available());
148             } else if (format.equals("INAM")) {
149                 this.name = chunk.readString(chunk.available());
150             } else if (format.equals("irom")) {
151                 this.romName = chunk.readString(chunk.available());
152             } else if (format.equals("iver")) {
153                 romVersionMajor = chunk.readUnsignedShort();
154                 romVersionMinor = chunk.readUnsignedShort();
155             } else if (format.equals("ICRD")) {
156                 this.creationDate = chunk.readString(chunk.available());
157             } else if (format.equals("IENG")) {
158                 this.engineers = chunk.readString(chunk.available());
159             } else if (format.equals("IPRD")) {
160                 this.product = chunk.readString(chunk.available());
161             } else if (format.equals("ICOP")) {
162                 this.copyright = chunk.readString(chunk.available());
163             } else if (format.equals("ICMT")) {
164                 this.comments = chunk.readString(chunk.available());
165             } else if (format.equals("ISFT")) {
166                 this.tools = chunk.readString(chunk.available());
167             }
168 
169         }
170     }
171 
172     private void readSdtaChunk(RIFFReader riff) throws IOException {
173         while (riff.hasNextChunk()) {
174             RIFFReader chunk = riff.nextChunk();
175             if (chunk.getFormat().equals("smpl")) {
176                 if (!largeFormat) {
177                     byte[] sampleData = new byte[chunk.available()];
178 
179                     int read = 0;
180                     int avail = chunk.available();
181                     while (read != avail) {
182                         if (avail - read > 65536) {
183                             chunk.readFully(sampleData, read, 65536);
184                             read += 65536;
```

```
                    } else {
                        chunk.readFully(sampleData, read, avail - read);
                        read = avail;
                    }

                }
                this.sampleData = new ModelByteBuffer(sampleData);
                //chunk.read(sampleData);
            } else {
                this.sampleData = new ModelByteBuffer(sampleFile,
                        chunk.getFilePointer(), chunk.available());
            }
        }
        if (chunk.getFormat().equals("sm24")) {
            if (!largeFormat) {
                byte[] sampleData24 = new byte[chunk.available()];
                //chunk.read(sampleData24);

                int read = 0;
                int avail = chunk.available();
                while (read != avail) {
                    if (avail - read > 65536) {
                        chunk.readFully(sampleData24, read, 65536);
                        read += 65536;
                    } else {
                        chunk.readFully(sampleData24, read, avail - read);
                        read = avail;
                    }

                }
                this.sampleData24 = new ModelByteBuffer(sampleData24);
            } else {
                this.sampleData24 = new ModelByteBuffer(sampleFile,
                        chunk.getFilePointer(), chunk.available());
            }

        }
    }
}

private void readPdtaChunk(RIFFReader riff) throws IOException {

    List<SF2Instrument> presets = new ArrayList<SF2Instrument>();
    List<Integer> presets_bagNdx = new ArrayList<Integer>();
    List<SF2InstrumentRegion> presets_splits_gen
            = new ArrayList<SF2InstrumentRegion>();
    List<SF2InstrumentRegion> presets_splits_mod
            = new ArrayList<SF2InstrumentRegion>();

    List<SF2Layer> instruments = new ArrayList<SF2Layer>();
    List<Integer> instruments_bagNdx = new ArrayList<Integer>();
    List<SF2LayerRegion> instruments_splits_gen
            = new ArrayList<SF2LayerRegion>();
    List<SF2LayerRegion> instruments_splits_mod
            = new ArrayList<SF2LayerRegion>();

    while (riff.hasNextChunk()) {
        RIFFReader chunk = riff.nextChunk();
        String format = chunk.getFormat();
        if (format.equals("phdr")) {
            // Preset Header / Instrument
            if (chunk.available() % 38 != 0)
```

```
                throw new RIFFInvalidDataException();
            int count = chunk.available() / 38;
            for (int i = 0; i < count; i++) {
                SF2Instrument preset = new SF2Instrument(this);
                preset.name = chunk.readString(20);
                preset.preset = chunk.readUnsignedShort();
                preset.bank = chunk.readUnsignedShort();
                presets_bagNdx.add(chunk.readUnsignedShort());
                preset.library = chunk.readUnsignedInt();
                preset.genre = chunk.readUnsignedInt();
                preset.morphology = chunk.readUnsignedInt();
                presets.add(preset);
                if (i != count - 1)
                    this.instruments.add(preset);
            }
        } else if (format.equals("pbag")) {
            // Preset Zones / Instruments splits
            if (chunk.available() % 4 != 0)
                throw new RIFFInvalidDataException();
            int count = chunk.available() / 4;

            // Skip first record
            {
                int gencount = chunk.readUnsignedShort();
                int modcount = chunk.readUnsignedShort();
                while (presets_splits_gen.size() < gencount)
                    presets_splits_gen.add(null);
                while (presets_splits_mod.size() < modcount)
                    presets_splits_mod.add(null);
                count--;
            }

            int offset = presets_bagNdx.get(0);
            // Offset should be 0 (but just case)
            for (int i = 0; i < offset; i++) {
                if (count == 0)
                    throw new RIFFInvalidDataException();
                int gencount = chunk.readUnsignedShort();
                int modcount = chunk.readUnsignedShort();
                while (presets_splits_gen.size() < gencount)
                    presets_splits_gen.add(null);
                while (presets_splits_mod.size() < modcount)
                    presets_splits_mod.add(null);
                count--;
            }

            for (int i = 0; i < presets_bagNdx.size() - 1; i++) {
                int zone_count = presets_bagNdx.get(i + 1)
                                 - presets_bagNdx.get(i);
                SF2Instrument preset = presets.get(i);
                for (int ii = 0; ii < zone_count; ii++) {
                    if (count == 0)
                        throw new RIFFInvalidDataException();
                    int gencount = chunk.readUnsignedShort();
                    int modcount = chunk.readUnsignedShort();
                    SF2InstrumentRegion split = new SF2InstrumentRegion();
                    preset.regions.add(split);
                    while (presets_splits_gen.size() < gencount)
                        presets_splits_gen.add(split);
                    while (presets_splits_mod.size() < modcount)
                        presets_splits_mod.add(split);
                    count--;
```

```
                }
            }
        } else if (format.equals("pmod")) {
            // Preset Modulators / Split Modulators
            for (int i = 0; i < presets_splits_mod.size(); i++) {
                SF2Modulator modulator = new SF2Modulator();
                modulator.sourceOperator = chunk.readUnsignedShort();
                modulator.destinationOperator = chunk.readUnsignedShort();
                modulator.amount = chunk.readShort();
                modulator.amountSourceOperator = chunk.readUnsignedShort();
                modulator.transportOperator = chunk.readUnsignedShort();
                SF2InstrumentRegion split = presets_splits_mod.get(i);
                if (split != null)
                    split.modulators.add(modulator);
            }
        } else if (format.equals("pgen")) {
            // Preset Generators / Split Generators
            for (int i = 0; i < presets_splits_gen.size(); i++) {
                int operator = chunk.readUnsignedShort();
                short amount = chunk.readShort();
                SF2InstrumentRegion split = presets_splits_gen.get(i);
                if (split != null)
                    split.generators.put(operator, amount);
            }
        } else if (format.equals("inst")) {
            // Instrument Header / Layers
            if (chunk.available() % 22 != 0)
                throw new RIFFInvalidDataException();
            int count = chunk.available() / 22;
            for (int i = 0; i < count; i++) {
                SF2Layer layer = new SF2Layer(this);
                layer.name = chunk.readString(20);
                instruments_bagNdx.add(chunk.readUnsignedShort());
                instruments.add(layer);
                if (i != count - 1)
                    this.layers.add(layer);
            }
        } else if (format.equals("ibag")) {
            // Instrument Zones / Layer splits
            if (chunk.available() % 4 != 0)
                throw new RIFFInvalidDataException();
            int count = chunk.available() / 4;

            // Skip first record
            {
                int gencount = chunk.readUnsignedShort();
                int modcount = chunk.readUnsignedShort();
                while (instruments_splits_gen.size() < gencount)
                    instruments_splits_gen.add(null);
                while (instruments_splits_mod.size() < modcount)
                    instruments_splits_mod.add(null);
                count--;
            }

            int offset = instruments_bagNdx.get(0);
            // Offset should be 0 (but just case)
            for (int i = 0; i < offset; i++) {
                if (count == 0)
                    throw new RIFFInvalidDataException();
                int gencount = chunk.readUnsignedShort();
                int modcount = chunk.readUnsignedShort();
                while (instruments_splits_gen.size() < gencount)
```

```
                    instruments_splits_gen.add(null);
                while (instruments_splits_mod.size() < modcount)
                    instruments_splits_mod.add(null);
                count--;
            }

            for (int i = 0; i < instruments_bagNdx.size() - 1; i++) {
                int zone_count = instruments_bagNdx.get(i + 1) - instruments_bagNdx.get(i);
                SF2Layer layer = layers.get(i);
                for (int ii = 0; ii < zone_count; ii++) {
                    if (count == 0)
                        throw new RIFFInvalidDataException();
                    int gencount = chunk.readUnsignedShort();
                    int modcount = chunk.readUnsignedShort();
                    SF2LayerRegion split = new SF2LayerRegion();
                    layer.regions.add(split);
                    while (instruments_splits_gen.size() < gencount)
                        instruments_splits_gen.add(split);
                    while (instruments_splits_mod.size() < modcount)
                        instruments_splits_mod.add(split);
                    count--;
                }
            }

        } else if (format.equals("imod")) {
            // Instrument Modulators / Split Modulators
            for (int i = 0; i < instruments_splits_mod.size(); i++) {
                SF2Modulator modulator = new SF2Modulator();
                modulator.sourceOperator = chunk.readUnsignedShort();
                modulator.destinationOperator = chunk.readUnsignedShort();
                modulator.amount = chunk.readShort();
                modulator.amountSourceOperator = chunk.readUnsignedShort();
                modulator.transportOperator = chunk.readUnsignedShort();
                SF2LayerRegion split = instruments_splits_gen.get(i);
                if (split != null)
                    split.modulators.add(modulator);
            }
        } else if (format.equals("igen")) {
            // Instrument Generators / Split Generators
            for (int i = 0; i < instruments_splits_gen.size(); i++) {
                int operator = chunk.readUnsignedShort();
                short amount = chunk.readShort();
                SF2LayerRegion split = instruments_splits_gen.get(i);
                if (split != null)
                    split.generators.put(operator, amount);
            }
        } else if (format.equals("shdr")) {
            // Sample Headers
            if (chunk.available() % 46 != 0)
                throw new RIFFInvalidDataException();
            int count = chunk.available() / 46;
            for (int i = 0; i < count; i++) {
                SF2Sample sample = new SF2Sample(this);
                sample.name = chunk.readString(20);
                long start = chunk.readUnsignedInt();
                long end = chunk.readUnsignedInt();
                sample.data = sampleData.subbuffer(start * 2, end * 2, true);
                if (sampleData24 != null)
                    sample.data24 = sampleData24.subbuffer(start, end, true);
                /*
                sample.data = new ModelByteBuffer(sampleData, (int)(start*2),
                        (int)((end - start)*2));
```

```
                if (sampleData24 != null)
                    sample.data24 = new ModelByteBuffer(sampleData24,
                            (int)start, (int)(end - start));
                 */
                sample.startLoop = chunk.readUnsignedInt() - start;
                sample.endLoop = chunk.readUnsignedInt() - start;
                if (sample.startLoop < 0)
                    sample.startLoop = -1;
                if (sample.endLoop < 0)
                    sample.endLoop = -1;
                sample.sampleRate = chunk.readUnsignedInt();
                sample.originalPitch = chunk.readUnsignedByte();
                sample.pitchCorrection = chunk.readByte();
                sample.sampleLink = chunk.readUnsignedShort();
                sample.sampleType = chunk.readUnsignedShort();
                if (i != count - 1)
                    this.samples.add(sample);
            }
        }
    }

    Iterator<SF2Layer> liter = this.layers.iterator();
    while (liter.hasNext()) {
        SF2Layer layer = liter.next();
        Iterator<SF2LayerRegion> siter = layer.regions.iterator();
        SF2Region globalsplit = null;
        while (siter.hasNext()) {
            SF2LayerRegion split = siter.next();
            if (split.generators.get(SF2LayerRegion.GENERATOR_SAMPLEID) != null) {
                int sampleid = split.generators.get(
                        SF2LayerRegion.GENERATOR_SAMPLEID);
                split.generators.remove(SF2LayerRegion.GENERATOR_SAMPLEID);
                split.sample = samples.get(sampleid);
            } else {
                globalsplit = split;
            }
        }
        if (globalsplit != null) {
            layer.getRegions().remove(globalsplit);
            SF2GlobalRegion gsplit = new SF2GlobalRegion();
            gsplit.generators = globalsplit.generators;
            gsplit.modulators = globalsplit.modulators;
            layer.setGlobalZone(gsplit);
        }
    }


    Iterator<SF2Instrument> iiter = this.instruments.iterator();
    while (iiter.hasNext()) {
        SF2Instrument instrument = iiter.next();
        Iterator<SF2InstrumentRegion> siter = instrument.regions.iterator();
        SF2Region globalsplit = null;
        while (siter.hasNext()) {
            SF2InstrumentRegion split = siter.next();
            if (split.generators.get(SF2LayerRegion.GENERATOR_INSTRUMENT) != null) {
                int instrumentid = split.generators.get(
                        SF2InstrumentRegion.GENERATOR_INSTRUMENT);
                split.generators.remove(SF2LayerRegion.GENERATOR_INSTRUMENT);
                split.layer = layers.get(instrumentid);
            } else {
                globalsplit = split;
            }
```

```
            }

            if (globalsplit != null) {
                instrument.getRegions().remove(globalsplit);
                SF2GlobalRegion gsplit = new SF2GlobalRegion();
                gsplit.generators = globalsplit.generators;
                gsplit.modulators = globalsplit.modulators;
                instrument.setGlobalZone(gsplit);
            }
        }

    }

    public void save(String name) throws IOException {
        writeSoundbank(new RIFFWriter(name, "sfbk"));
    }

    public void save(File file) throws IOException {
        writeSoundbank(new RIFFWriter(file, "sfbk"));
    }

    public void save(OutputStream out) throws IOException {
        writeSoundbank(new RIFFWriter(out, "sfbk"));
    }

    private void writeSoundbank(RIFFWriter writer) throws IOException {
        writeInfo(writer.writeList("INFO"));
        writeSdtaChunk(writer.writeList("sdta"));
        writePdtaChunk(writer.writeList("pdta"));
        writer.close();
    }

    private void writeInfoStringChunk(RIFFWriter writer, String name,
            String value) throws IOException {
        if (value == null)
            return;
        RIFFWriter chunk = writer.writeChunk(name);
        chunk.writeString(value);
        int len = value.getBytes("ascii").length;
        chunk.write(0);
        len++;
        if (len % 2 != 0)
            chunk.write(0);
    }

    private void writeInfo(RIFFWriter writer) throws IOException {
        if (this.targetEngine == null)
            this.targetEngine = "EMU8000";
        if (this.name == null)
            this.name = "";

        RIFFWriter ifil_chunk = writer.writeChunk("ifil");
        ifil_chunk.writeUnsignedShort(this.major);
        ifil_chunk.writeUnsignedShort(this.minor);
        writeInfoStringChunk(writer, "isng", this.targetEngine);
        writeInfoStringChunk(writer, "INAM", this.name);
        writeInfoStringChunk(writer, "irom", this.romName);
        if (romVersionMajor != -1) {
            RIFFWriter iver_chunk = writer.writeChunk("iver");
            iver_chunk.writeUnsignedShort(this.romVersionMajor);
            iver_chunk.writeUnsignedShort(this.romVersionMinor);
        }
```

```
        writeInfoStringChunk(writer, "ICRD", this.creationDate);
        writeInfoStringChunk(writer, "IENG", this.engineers);
        writeInfoStringChunk(writer, "IPRD", this.product);
        writeInfoStringChunk(writer, "ICOP", this.copyright);
        writeInfoStringChunk(writer, "ICMT", this.comments);
        writeInfoStringChunk(writer, "ISFT", this.tools);

        writer.close();
    }

    private void writeSdtaChunk(RIFFWriter writer) throws IOException {

        byte[] pad = new byte[32];

        RIFFWriter smpl_chunk = writer.writeChunk("smpl");
        for (SF2Sample sample : samples) {
            ModelByteBuffer data = sample.getDataBuffer();
            data.writeTo(smpl_chunk);
            /*
            smpl_chunk.write(data.array(),
            data.arrayOffset(),
            data.capacity());
             */
            smpl_chunk.write(pad);
            smpl_chunk.write(pad);
        }
        if (major < 2)
            return;
        if (major == 2 && minor < 4)
            return;


        for (SF2Sample sample : samples) {
            ModelByteBuffer data24 = sample.getData24Buffer();
            if (data24 == null)
                return;
        }

        RIFFWriter sm24_chunk = writer.writeChunk("sm24");
        for (SF2Sample sample : samples) {
            ModelByteBuffer data = sample.getData24Buffer();
            data.writeTo(sm24_chunk);
            /*
            sm24_chunk.write(data.array(),
            data.arrayOffset(),
            data.capacity());*/
            smpl_chunk.write(pad);
        }
    }

    private void writeModulators(RIFFWriter writer, List<SF2Modulator> modulators)
            throws IOException {
        for (SF2Modulator modulator : modulators) {
            writer.writeUnsignedShort(modulator.sourceOperator);
            writer.writeUnsignedShort(modulator.destinationOperator);
            writer.writeShort(modulator.amount);
            writer.writeUnsignedShort(modulator.amountSourceOperator);
            writer.writeUnsignedShort(modulator.transportOperator);
        }
    }

    private void writeGenerators(RIFFWriter writer, Map<Integer, Short> generators)
```

```
            throws IOException {
        Short keyrange = (Short) generators.get(SF2Region.GENERATOR_KEYRANGE);
        Short velrange = (Short) generators.get(SF2Region.GENERATOR_VELRANGE);
        if (keyrange != null) {
            writer.writeUnsignedShort(SF2Region.GENERATOR_KEYRANGE);
            writer.writeShort(keyrange);
        }
        if (velrange != null) {
            writer.writeUnsignedShort(SF2Region.GENERATOR_VELRANGE);
            writer.writeShort(velrange);
        }
        for (Map.Entry<Integer, Short> generator : generators.entrySet()) {
            if (generator.getKey() == SF2Region.GENERATOR_KEYRANGE)
                continue;
            if (generator.getKey() == SF2Region.GENERATOR_VELRANGE)
                continue;
            writer.writeUnsignedShort(generator.getKey());
            writer.writeShort(generator.getValue());
        }
    }

    private void writePdtaChunk(RIFFWriter writer) throws IOException {

        RIFFWriter phdr_chunk = writer.writeChunk("phdr");
        int phdr_zone_count = 0;
        for (SF2Instrument preset : this.instruments) {
            phdr_chunk.writeString(preset.name, 20);
            phdr_chunk.writeUnsignedShort(preset.preset);
            phdr_chunk.writeUnsignedShort(preset.bank);
            phdr_chunk.writeUnsignedShort(phdr_zone_count);
            if (preset.getGlobalRegion() != null)
                phdr_zone_count += 1;
            phdr_zone_count += preset.getRegions().size();
            phdr_chunk.writeUnsignedInt(preset.library);
            phdr_chunk.writeUnsignedInt(preset.genre);
            phdr_chunk.writeUnsignedInt(preset.morphology);
        }
        phdr_chunk.writeString("EOP", 20);
        phdr_chunk.writeUnsignedShort(0);
        phdr_chunk.writeUnsignedShort(0);
        phdr_chunk.writeUnsignedShort(phdr_zone_count);
        phdr_chunk.writeUnsignedInt(0);
        phdr_chunk.writeUnsignedInt(0);
        phdr_chunk.writeUnsignedInt(0);


        RIFFWriter pbag_chunk = writer.writeChunk("pbag");
        int pbag_gencount = 0;
        int pbag_modcount = 0;
        for (SF2Instrument preset : this.instruments) {
            if (preset.getGlobalRegion() != null) {
                pbag_chunk.writeUnsignedShort(pbag_gencount);
                pbag_chunk.writeUnsignedShort(pbag_modcount);
                pbag_gencount += preset.getGlobalRegion().getGenerators().size();
                pbag_modcount += preset.getGlobalRegion().getModulators().size();
            }
            for (SF2InstrumentRegion region : preset.getRegions()) {
                pbag_chunk.writeUnsignedShort(pbag_gencount);
                pbag_chunk.writeUnsignedShort(pbag_modcount);
                if (layers.indexOf(region.layer) != -1) {
                    // One generator is used to reference to instrument record
                    pbag_gencount += 1;
```

```
            }
            pbag_gencount += region.getGenerators().size();
            pbag_modcount += region.getModulators().size();

        }
    }
    pbag_chunk.writeUnsignedShort(pbag_gencount);
    pbag_chunk.writeUnsignedShort(pbag_modcount);

    RIFFWriter pmod_chunk = writer.writeChunk("pmod");
    for (SF2Instrument preset : this.instruments) {
        if (preset.getGlobalRegion() != null) {
            writeModulators(pmod_chunk,
                    preset.getGlobalRegion().getModulators());
        }
        for (SF2InstrumentRegion region : preset.getRegions())
            writeModulators(pmod_chunk, region.getModulators());
    }
    pmod_chunk.write(new byte[10]);

    RIFFWriter pgen_chunk = writer.writeChunk("pgen");
    for (SF2Instrument preset : this.instruments) {
        if (preset.getGlobalRegion() != null) {
            writeGenerators(pgen_chunk,
                    preset.getGlobalRegion().getGenerators());
        }
        for (SF2InstrumentRegion region : preset.getRegions()) {
            writeGenerators(pgen_chunk, region.getGenerators());
            int ix = (int) layers.indexOf(region.layer);
            if (ix != -1) {
                pgen_chunk.writeUnsignedShort(SF2Region.GENERATOR_INSTRUMENT);
                pgen_chunk.writeShort((short) ix);
            }
        }
    }
    pgen_chunk.write(new byte[4]);

    RIFFWriter inst_chunk = writer.writeChunk("inst");
    int inst_zone_count = 0;
    for (SF2Layer instrument : this.layers) {
        inst_chunk.writeString(instrument.name, 20);
        inst_chunk.writeUnsignedShort(inst_zone_count);
        if (instrument.getGlobalRegion() != null)
            inst_zone_count += 1;
        inst_zone_count += instrument.getRegions().size();
    }
    inst_chunk.writeString("EOI", 20);
    inst_chunk.writeUnsignedShort(inst_zone_count);


    RIFFWriter ibag_chunk = writer.writeChunk("ibag");
    int ibag_gencount = 0;
    int ibag_modcount = 0;
    for (SF2Layer instrument : this.layers) {
        if (instrument.getGlobalRegion() != null) {
            ibag_chunk.writeUnsignedShort(ibag_gencount);
            ibag_chunk.writeUnsignedShort(ibag_modcount);
            ibag_gencount
                    += instrument.getGlobalRegion().getGenerators().size();
            ibag_modcount
                    += instrument.getGlobalRegion().getModulators().size();
        }
```

```
            for (SF2LayerRegion region : instrument.getRegions()) {
                ibag_chunk.writeUnsignedShort(ibag_gencount);
                ibag_chunk.writeUnsignedShort(ibag_modcount);
                if (samples.indexOf(region.sample) != -1) {
                    // One generator is used to reference to instrument record
                    ibag_gencount += 1;
                }
                ibag_gencount += region.getGenerators().size();
                ibag_modcount += region.getModulators().size();

            }
        }
        ibag_chunk.writeUnsignedShort(ibag_gencount);
        ibag_chunk.writeUnsignedShort(ibag_modcount);


        RIFFWriter imod_chunk = writer.writeChunk("imod");
        for (SF2Layer instrument : this.layers) {
            if (instrument.getGlobalRegion() != null) {
                writeModulators(imod_chunk,
                        instrument.getGlobalRegion().getModulators());
            }
            for (SF2LayerRegion region : instrument.getRegions())
                writeModulators(imod_chunk, region.getModulators());
        }
        imod_chunk.write(new byte[10]);

        RIFFWriter igen_chunk = writer.writeChunk("igen");
        for (SF2Layer instrument : this.layers) {
            if (instrument.getGlobalRegion() != null) {
                writeGenerators(igen_chunk,
                        instrument.getGlobalRegion().getGenerators());
            }
            for (SF2LayerRegion region : instrument.getRegions()) {
                writeGenerators(igen_chunk, region.getGenerators());
                int ix = samples.indexOf(region.sample);
                if (ix != -1) {
                    igen_chunk.writeUnsignedShort(SF2Region.GENERATOR_SAMPLEID);
                    igen_chunk.writeShort((short) ix);
                }
            }
        }
        igen_chunk.write(new byte[4]);


        RIFFWriter shdr_chunk = writer.writeChunk("shdr");
        long sample_pos = 0;
        for (SF2Sample sample : samples) {
            shdr_chunk.writeString(sample.name, 20);
            long start = sample_pos;
            sample_pos += sample.data.capacity() / 2;
            long end = sample_pos;
            long startLoop = sample.startLoop + start;
            long endLoop = sample.endLoop + start;
            if (startLoop < start)
                startLoop = start;
            if (endLoop > end)
                endLoop = end;
            shdr_chunk.writeUnsignedInt(start);
            shdr_chunk.writeUnsignedInt(end);
            shdr_chunk.writeUnsignedInt(startLoop);
            shdr_chunk.writeUnsignedInt(endLoop);
```

```
            shdr_chunk.writeUnsignedInt(sample.sampleRate);
            shdr_chunk.writeUnsignedByte(sample.originalPitch);
            shdr_chunk.writeByte(sample.pitchCorrection);
            shdr_chunk.writeUnsignedShort(sample.sampleLink);
            shdr_chunk.writeUnsignedShort(sample.sampleType);
            sample_pos += 32;
        }
        shdr_chunk.writeString("EOS", 20);
        shdr_chunk.write(new byte[26]);

    }

    public String getName() {
        return name;
    }

    public String getVersion() {
        return major + "." + minor;
    }

    public String getVendor() {
        return engineers;
    }

    public String getDescription() {
        return comments;
    }

    public void setName(String s) {
        name = s;
    }

    public void setVendor(String s) {
        engineers = s;
    }

    public void setDescription(String s) {
        comments = s;
    }

    public SoundbankResource[] getResources() {
        SoundbankResource[] resources
                = new SoundbankResource[layers.size() + samples.size()];
        int j = 0;
        for (int i = 0; i < layers.size(); i++)
            resources[j++] = layers.get(i);
        for (int i = 0; i < samples.size(); i++)
            resources[j++] = samples.get(i);
        return resources;
    }

    public SF2Instrument[] getInstruments() {
        SF2Instrument[] inslist_array
                = instruments.toArray(new SF2Instrument[instruments.size()]);
        Arrays.sort(inslist_array, new ModelInstrumentComparator());
        return inslist_array;
    }

    public SF2Layer[] getLayers() {
        return layers.toArray(new SF2Layer[layers.size()]);
    }

```

```
    public SF2Sample[] getSamples() {
        return samples.toArray(new SF2Sample[samples.size()]);
    }

    public Instrument getInstrument(Patch patch) {
        int program = patch.getProgram();
        int bank = patch.getBank();
        boolean percussion = false;
        if (patch instanceof ModelPatch)
            percussion = ((ModelPatch)patch).isPercussion();
        for (Instrument instrument : instruments) {
            Patch patch2 = instrument.getPatch();
            int program2 = patch2.getProgram();
            int bank2 = patch2.getBank();
            if (program == program2 && bank == bank2) {
                boolean percussion2 = false;
                if (patch2 instanceof ModelPatch)
                    percussion2 = ((ModelPatch) patch2).isPercussion();
                if (percussion == percussion2)
                    return instrument;
            }
        }
        return null;
    }

    public String getCreationDate() {
        return creationDate;
    }

    public void setCreationDate(String creationDate) {
        this.creationDate = creationDate;
    }

    public String getProduct() {
        return product;
    }

    public void setProduct(String product) {
        this.product = product;
    }

    public String getRomName() {
        return romName;
    }

    public void setRomName(String romName) {
        this.romName = romName;
    }

    public int getRomVersionMajor() {
        return romVersionMajor;
    }

    public void setRomVersionMajor(int romVersionMajor) {
        this.romVersionMajor = romVersionMajor;
    }

    public int getRomVersionMinor() {
        return romVersionMinor;
    }

    public void setRomVersionMinor(int romVersionMinor) {
```

```
929         this.romVersionMinor = romVersionMinor;
930     }
931 
932     public String getTargetEngine() {
933         return targetEngine;
934     }
935 
936     public void setTargetEngine(String targetEngine) {
937         this.targetEngine = targetEngine;
938     }
939 
940     public String getTools() {
941         return tools;
942     }
943 
944     public void setTools(String tools) {
945         this.tools = tools;
946     }
947 
948     public void addResource(SoundbankResource resource) {
949         if (resource instanceof SF2Instrument)
950             instruments.add((SF2Instrument)resource);
951         if (resource instanceof SF2Layer)
952             layers.add((SF2Layer)resource);
953         if (resource instanceof SF2Sample)
954             samples.add((SF2Sample)resource);
955     }
956 
957     public void removeResource(SoundbankResource resource) {
958         if (resource instanceof SF2Instrument)
959             instruments.remove((SF2Instrument)resource);
960         if (resource instanceof SF2Layer)
961             layers.remove((SF2Layer)resource);
962         if (resource instanceof SF2Sample)
963             samples.remove((SF2Sample)resource);
964     }
965 
966     public void addInstrument(SF2Instrument resource) {
967         instruments.add(resource);
968     }
969 
970     public void removeInstrument(SF2Instrument resource) {
971         instruments.remove(resource);
972     }
973 }
```

74 com/sun/media/sound/SF2SoundbankReader.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import javax.sound.midi.InvalidMidiDataException;
import javax.sound.midi.Soundbank;
import javax.sound.midi.spi.SoundbankReader;

/**
 * This class is used to connect the SF2SoundBank class
 * to the SoundbankReader SPI interface.
 *
 * @author Karl Helgason
 */
public class SF2SoundbankReader extends SoundbankReader {

    public Soundbank getSoundbank(URL url)
            throws InvalidMidiDataException, IOException {
        try {
            return new SF2Soundbank(url);
        } catch (RIFFInvalidFormatException e) {
            return null;
        } catch(IOException ioe) {
            return null;
        }
    }

    public Soundbank getSoundbank(InputStream stream)
            throws InvalidMidiDataException, IOException {
        try {
            stream.mark(512);
            return new SF2Soundbank(stream);
        } catch (RIFFInvalidFormatException e) {
            stream.reset();
```

```
            return null;
        }
    }

    public Soundbank getSoundbank(File file)
            throws InvalidMidiDataException, IOException {
        try {
            return new SF2Soundbank(file);
        } catch (RIFFInvalidFormatException e) {
            return null;
        }
    }
}
```

75 com/sun/media/sound/SimpleInstrument.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.util.ArrayList;
import java.util.List;
import javax.sound.midi.Patch;

/**
 * A simple instrument that is made of other ModelInstrument, ModelPerformer
 * objects.
 *
 * @author Karl Helgason
 */
public class SimpleInstrument extends ModelInstrument {

    private static class SimpleInstrumentPart {
        ModelPerformer[] performers;
        int keyFrom;
        int keyTo;
        int velFrom;
        int velTo;
        int exclusiveClass;
    }
    protected int preset = 0;
    protected int bank = 0;
    protected boolean percussion = false;
    protected String name = "";
    protected List<SimpleInstrumentPart> parts
            = new ArrayList<SimpleInstrumentPart>();

    public SimpleInstrument() {
        super(null, null, null, null);
    }

    public void clear() {
        parts.clear();
    }
```

```
61
62      public void add(ModelPerformer[] performers, int keyFrom, int keyTo,
63              int velFrom, int velTo, int exclusiveClass) {
64          SimpleInstrumentPart part = new SimpleInstrumentPart();
65          part.performers = performers;
66          part.keyFrom = keyFrom;
67          part.keyTo = keyTo;
68          part.velFrom = velFrom;
69          part.velTo = velTo;
70          part.exclusiveClass = exclusiveClass;
71          parts.add(part);
72      }
73
74      public void add(ModelPerformer[] performers, int keyFrom, int keyTo,
75              int velFrom, int velTo) {
76          add(performers, keyFrom, keyTo, velFrom, velTo, -1);
77      }
78
79      public void add(ModelPerformer[] performers, int keyFrom, int keyTo) {
80          add(performers, keyFrom, keyTo, 0, 127, -1);
81      }
82
83      public void add(ModelPerformer[] performers) {
84          add(performers, 0, 127, 0, 127, -1);
85      }
86
87      public void add(ModelPerformer performer, int keyFrom, int keyTo,
88              int velFrom, int velTo, int exclusiveClass) {
89          add(new ModelPerformer[]{performer}, keyFrom, keyTo, velFrom, velTo,
90                  exclusiveClass);
91      }
92
93      public void add(ModelPerformer performer, int keyFrom, int keyTo,
94              int velFrom, int velTo) {
95          add(new ModelPerformer[]{performer}, keyFrom, keyTo, velFrom, velTo);
96      }
97
98      public void add(ModelPerformer performer, int keyFrom, int keyTo) {
99          add(new ModelPerformer[]{performer}, keyFrom, keyTo);
100     }
101
102     public void add(ModelPerformer performer) {
103         add(new ModelPerformer[]{performer});
104     }
105
106     public void add(ModelInstrument ins, int keyFrom, int keyTo, int velFrom,
107             int velTo, int exclusiveClass) {
108         add(ins.getPerformers(), keyFrom, keyTo, velFrom, velTo, exclusiveClass);
109     }
110
111     public void add(ModelInstrument ins, int keyFrom, int keyTo, int velFrom,
112             int velTo) {
113         add(ins.getPerformers(), keyFrom, keyTo, velFrom, velTo);
114     }
115
116     public void add(ModelInstrument ins, int keyFrom, int keyTo) {
117         add(ins.getPerformers(), keyFrom, keyTo);
118     }
119
120     public void add(ModelInstrument ins) {
121         add(ins.getPerformers());
122     }
```

```
123
124     public ModelPerformer[] getPerformers() {
125
126         int percount = 0;
127         for (SimpleInstrumentPart part : parts)
128             if (part.performers != null)
129                 percount += part.performers.length;
130
131         ModelPerformer[] performers = new ModelPerformer[percount];
132         int px = 0;
133         for (SimpleInstrumentPart part : parts) {
134             if (part.performers != null) {
135                 for (ModelPerformer mperfm : part.performers) {
136                     ModelPerformer performer = new ModelPerformer();
137                     performer.setName(getName());
138                     performers[px++] = performer;
139
140                     performer.setDefaultConnectionsEnabled(
141                             mperfm.isDefaultConnectionsEnabled());
142                     performer.setKeyFrom(mperfm.getKeyFrom());
143                     performer.setKeyTo(mperfm.getKeyTo());
144                     performer.setVelFrom(mperfm.getVelFrom());
145                     performer.setVelTo(mperfm.getVelTo());
146                     performer.setExclusiveClass(mperfm.getExclusiveClass());
147                     performer.setSelfNonExclusive(mperfm.isSelfNonExclusive());
148                     performer.setReleaseTriggered(mperfm.isReleaseTriggered());
149                     if (part.exclusiveClass != -1)
150                         performer.setExclusiveClass(part.exclusiveClass);
151                     if (part.keyFrom > performer.getKeyFrom())
152                         performer.setKeyFrom(part.keyFrom);
153                     if (part.keyTo < performer.getKeyTo())
154                         performer.setKeyTo(part.keyTo);
155                     if (part.velFrom > performer.getVelFrom())
156                         performer.setVelFrom(part.velFrom);
157                     if (part.velTo < performer.getVelTo())
158                         performer.setVelTo(part.velTo);
159                     performer.getOscillators().addAll(mperfm.getOscillators());
160                     performer.getConnectionBlocks().addAll(
161                             mperfm.getConnectionBlocks());
162                 }
163             }
164         }
165
166         return performers;
167     }
168
169     public Object getData() {
170         return null;
171     }
172
173     public String getName() {
174         return this.name;
175     }
176
177     public void setName(String name) {
178         this.name = name;
179     }
180
181     public ModelPatch getPatch() {
182         return new ModelPatch(bank, preset, percussion);
183     }
184
```

```
    public void setPatch(Patch patch) {
        if (patch instanceof ModelPatch && ((ModelPatch)patch).isPercussion()) {
            percussion = true;
            bank = patch.getBank();
            preset = patch.getProgram();
        } else {
            percussion = false;
            bank = patch.getBank();
            preset = patch.getProgram();
        }
    }
}
```

76 com/sun/media/sound/SimpleSoundbank.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

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

import javax.sound.midi.Instrument;
import javax.sound.midi.Patch;
import javax.sound.midi.Soundbank;
import javax.sound.midi.SoundbankResource;

/**
 * A simple soundbank that contains instruments and soundbankresources.
 *
 * @author Karl Helgason
 */
public class SimpleSoundbank implements Soundbank {

    String name = "";
    String version = "";
    String vendor = "";
    String description = "";
    List<SoundbankResource> resources = new ArrayList<SoundbankResource>();
    List<Instrument> instruments = new ArrayList<Instrument>();

    public String getName() {
        return name;
    }

    public String getVersion() {
        return version;
    }

    public String getVendor() {
        return vendor;
    }
```

```
61
62      public String getDescription() {
63          return description;
64      }
65
66      public void setDescription(String description) {
67          this.description = description;
68      }
69
70      public void setName(String name) {
71          this.name = name;
72      }
73
74      public void setVendor(String vendor) {
75          this.vendor = vendor;
76      }
77
78      public void setVersion(String version) {
79          this.version = version;
80      }
81
82      public SoundbankResource[] getResources() {
83          return resources.toArray(new SoundbankResource[resources.size()]);
84      }
85
86      public Instrument[] getInstruments() {
87          Instrument[] inslist_array
88                  = instruments.toArray(new Instrument[resources.size()]);
89          Arrays.sort(inslist_array, new ModelInstrumentComparator());
90          return inslist_array;
91      }
92
93      public Instrument getInstrument(Patch patch) {
94          int program = patch.getProgram();
95          int bank = patch.getBank();
96          boolean percussion = false;
97          if (patch instanceof ModelPatch)
98              percussion = ((ModelPatch)patch).isPercussion();
99          for (Instrument instrument : instruments) {
100             Patch patch2 = instrument.getPatch();
101             int program2 = patch2.getProgram();
102             int bank2 = patch2.getBank();
103             if (program == program2 && bank == bank2) {
104                 boolean percussion2 = false;
105                 if (patch2 instanceof ModelPatch)
106                     percussion2 = ((ModelPatch)patch2).isPercussion();
107                 if (percussion == percussion2)
108                     return instrument;
109             }
110         }
111         return null;
112     }
113
114     public void addResource(SoundbankResource resource) {
115         if (resource instanceof Instrument)
116             instruments.add((Instrument) resource);
117         else
118             resources.add(resource);
119     }
120
121     public void removeResource(SoundbankResource resource) {
122         if (resource instanceof Instrument)
```

```
            instruments.remove((Instrument) resource);
        else
            resources.remove(resource);
    }

    public void addInstrument(Instrument resource) {
        instruments.add(resource);
    }

    public void removeInstrument(Instrument resource) {
        instruments.remove(resource);
    }

    public void addAllInstruments(Soundbank soundbank) {
        for (Instrument ins : soundbank.getInstruments())
            addInstrument(ins);
    }

    public void removeAllInstruments(Soundbank soundbank) {
        for (Instrument ins : soundbank.getInstruments())
            removeInstrument(ins);
    }
}
```

77 com/sun/media/sound/SoftAbstractResampler.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.io.IOException;
import java.util.Arrays;

import javax.sound.midi.MidiChannel;
import javax.sound.midi.VoiceStatus;

/**
 * Abstract resampler class.
 *
 * @author Karl Helgason
 */
public abstract class SoftAbstractResampler implements SoftResampler {

    private class ModelAbstractResamplerStream implements SoftResamplerStreamer {

        AudioFloatInputStream stream;
        boolean stream_eof = false;
        int loopmode;
        boolean loopdirection = true; // true = forward
        float loopstart;
        float looplen;
        float target_pitch;
        float[] current_pitch = new float[1];
        boolean started;
        boolean eof;
        int sector_pos = 0;
        int sector_size = 400;
        int sector_loopstart = -1;
        boolean markset = false;
        int marklimit = 0;
        int streampos = 0;
        int nrofchannels = 2;
        boolean noteOff_flag = false;
        float[][] ibuffer;
```

```
        boolean ibuffer_order = true;
        float[] sbuffer;
        int pad;
        int pad2;
        float[] ix = new float[1];
        int[] ox = new int[1];
        float samplerateconv = 1;
        float pitchcorrection = 0;

        public ModelAbstractResamplerStream() {
            pad = getPadding();
            pad2 = getPadding() * 2;
            ibuffer = new float[2][sector_size + pad2];
            ibuffer_order = true;
        }

        public void noteOn(MidiChannel channel, VoiceStatus voice,
                int noteNumber, int velocity) {
        }

        public void noteOff(int velocity) {
            noteOff_flag = true;
        }

        public void open(ModelWavetable osc, float outputsamplerate)
                throws IOException {

            eof = false;
            nrofchannels = osc.getChannels();
            if (ibuffer.length < nrofchannels) {
                ibuffer = new float[nrofchannels][sector_size + pad2];
            }

            stream = osc.openStream();
            streampos = 0;
            stream_eof = false;
            pitchcorrection = osc.getPitchcorrection();
            samplerateconv
                    = stream.getFormat().getSampleRate() / outputsamplerate;
            looplen = osc.getLoopLength();
            loopstart = osc.getLoopStart();
            sector_loopstart = (int) (loopstart / sector_size);
            sector_loopstart = sector_loopstart - 1;

            sector_pos = 0;

            if (sector_loopstart < 0)
                sector_loopstart = 0;
            started = false;
            loopmode = osc.getLoopType();

            if (loopmode != 0) {
                markset = false;
                marklimit = nrofchannels * (int) (looplen + pad2 + 1);
            } else
                markset = true;
            // loopmode = 0;

            target_pitch = samplerateconv;
            current_pitch[0] = samplerateconv;

            ibuffer_order = true;
```

```
            loopdirection = true;
            noteOff_flag = false;

            for (int i = 0; i < nrofchannels; i++)
                Arrays.fill(ibuffer[i], sector_size, sector_size + pad2, 0);
            ix[0] = pad;
            eof = false;

            ix[0] = sector_size + pad;
            sector_pos = -1;
            streampos = -sector_size;

            nextBuffer();
        }

        public void setPitch(float pitch) {
            /*
            this.pitch = (float) Math.pow(2f,
            (pitchcorrection + pitch) / 1200.0f)
             * samplerateconv;
             */
            this.target_pitch = (float)Math.exp(
                    (pitchcorrection + pitch) * (Math.log(2.0) / 1200.0))
                * samplerateconv;

            if (!started)
                current_pitch[0] = this.target_pitch;
        }

        public void nextBuffer() throws IOException {
            if (ix[0] < pad) {
                if (markset) {
                    // reset to target sector
                    stream.reset();
                    ix[0] += streampos - (sector_loopstart * sector_size);
                    sector_pos = sector_loopstart;
                    streampos = sector_pos * sector_size;

                    // and go one sector backward
                    ix[0] += sector_size;
                    sector_pos -= 1;
                    streampos -= sector_size;
                    stream_eof = false;
                }
            }

            if (ix[0] >= sector_size + pad) {
                if (stream_eof) {
                    eof = true;
                    return;
                }
            }

            if (ix[0] >= sector_size * 4 + pad) {
                int skips = (int)((ix[0] - sector_size * 4 + pad) / sector_size);
                ix[0] -= sector_size * skips;
                sector_pos += skips;
                streampos += sector_size * skips;
                stream.skip(sector_size * skips);
            }

            while (ix[0] >= sector_size + pad) {
```

```
            if (!markset) {
                if (sector_pos + 1 == sector_loopstart) {
                    stream.mark(marklimit);
                    markset = true;
                }
            }
            ix[0] -= sector_size;
            sector_pos++;
            streampos += sector_size;

            for (int c = 0; c < nrofchannels; c++) {
                float[] cbuffer = ibuffer[c];
                for (int i = 0; i < pad2; i++)
                    cbuffer[i] = cbuffer[i + sector_size];
            }

            int ret;
            if (nrofchannels == 1)
                ret = stream.read(ibuffer[0], pad2, sector_size);
            else {
                int slen = sector_size * nrofchannels;
                if (sbuffer == null || sbuffer.length < slen)
                    sbuffer = new float[slen];
                int sret = stream.read(sbuffer, 0, slen);
                if (sret == -1)
                    ret = -1;
                else {
                    ret = sret / nrofchannels;
                    for (int i = 0; i < nrofchannels; i++) {
                        float[] buff = ibuffer[i];
                        int ix = i;
                        int ix_step = nrofchannels;
                        int ox = pad2;
                        for (int j = 0; j < ret; j++, ix += ix_step, ox++)
                            buff[ox] = sbuffer[ix];
                    }
                }

            }

            if (ret == -1) {
                ret = 0;
                stream_eof = true;
                for (int i = 0; i < nrofchannels; i++)
                    Arrays.fill(ibuffer[i], pad2, pad2 + sector_size, 0f);
                return;
            }
            if (ret != sector_size) {
                for (int i = 0; i < nrofchannels; i++)
                    Arrays.fill(ibuffer[i], pad2 + ret, pad2 + sector_size, 0f);
            }

            ibuffer_order = true;

        }

    }

    public void reverseBuffers() {
        ibuffer_order = !ibuffer_order;
        for (int c = 0; c < nrofchannels; c++) {
            float[] cbuff = ibuffer[c];
```

```
            int len = cbuff.length - 1;
            int len2 = cbuff.length / 2;
            for (int i = 0; i < len2; i++) {
                float x = cbuff[i];
                cbuff[i] = cbuff[len - i];
                cbuff[len - i] = x;
            }
        }
    }

    public int read(float[][] buffer, int offset, int len)
            throws IOException {

        if (eof)
            return -1;

        if (noteOff_flag)
            if ((loopmode & 2) != 0)
                if (loopdirection)
                    loopmode = 0;


        float pitchstep = (target_pitch - current_pitch[0]) / len;
        float[] current_pitch = this.current_pitch;
        started = true;

        int[] ox = this.ox;
        ox[0] = offset;
        int ox_end = len + offset;

        float ixend = sector_size + pad;
        if (!loopdirection)
            ixend = pad;
        while (ox[0] != ox_end) {
            nextBuffer();
            if (!loopdirection) {
                // If we are in backward playing part of pingpong
                // or reverse loop

                if (streampos < (loopstart + pad)) {
                    ixend = loopstart - streampos + pad2;
                    if (ix[0] <= ixend) {
                        if ((loopmode & 4) != 0) {
                            // Ping pong loop, change loopdirection
                            loopdirection = true;
                            ixend = sector_size + pad;
                            continue;
                        }

                        ix[0] += looplen;
                        ixend = pad;
                        continue;
                    }
                }

                if (ibuffer_order != loopdirection)
                    reverseBuffers();

                ix[0] = (sector_size + pad2) - ix[0];
                ixend = (sector_size + pad2) - ixend;
                ixend++;
```

```
                float bak_ix = ix[0];
                int bak_ox = ox[0];
                float bak_pitch = current_pitch[0];
                for (int i = 0; i < nrofchannels; i++) {
                    if (buffer[i] != null) {
                        ix[0] = bak_ix;
                        ox[0] = bak_ox;
                        current_pitch[0] = bak_pitch;
                        interpolate(ibuffer[i], ix, ixend, current_pitch,
                                pitchstep, buffer[i], ox, ox_end);
                    }
                }

                ix[0] = (sector_size + pad2) - ix[0];
                ixend--;
                ixend = (sector_size + pad2) - ixend;

                if (eof) {
                    current_pitch[0] = this.target_pitch;
                    return ox[0] - offset;
                }

                continue;
            }
            if (loopmode != 0) {
                if (streampos + sector_size > (looplen + loopstart + pad)) {
                    ixend = loopstart + looplen - streampos + pad2;
                    if (ix[0] >= ixend) {
                        if ((loopmode & 4) != 0 || (loopmode & 8) != 0) {
                            // Ping pong or revese loop, change loopdirection
                            loopdirection = false;
                            ixend = pad;
                            continue;
                        }
                        ixend = sector_size + pad;
                        ix[0] -= looplen;
                        continue;
                    }
                }
            }

            if (ibuffer_order != loopdirection)
                reverseBuffers();

            float bak_ix = ix[0];
            int bak_ox = ox[0];
            float bak_pitch = current_pitch[0];
            for (int i = 0; i < nrofchannels; i++) {
                if (buffer[i] != null) {
                    ix[0] = bak_ix;
                    ox[0] = bak_ox;
                    current_pitch[0] = bak_pitch;
                    interpolate(ibuffer[i], ix, ixend, current_pitch,
                            pitchstep, buffer[i], ox, ox_end);
                }
            }

            if (eof) {
                current_pitch[0] = this.target_pitch;
                return ox[0] - offset;
            }
        }
```

```
371
372             current_pitch[0] = this.target_pitch;
373             return len;
374         }
375
376         public void close() throws IOException {
377             stream.close();
378         }
379     }
380
381     public abstract int getPadding();
382
383     public abstract void interpolate(float[] in, float[] in_offset,
384             float in_end, float[] pitch, float pitchstep, float[] out,
385             int[] out_offset, int out_end);
386
387     public SoftResamplerStreamer openStreamer() {
388         return new ModelAbstractResamplerStream();
389     }
390 }
```

78 com/sun/media/sound/SoftAudioBuffer.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.util.Arrays;

import javax.sound.sampled.AudioFormat;

/**
 * This class is used to store audio buffer.
 *
 * @author Karl Helgason
 */
public class SoftAudioBuffer {

    private int size;
    private float[] buffer;
    private boolean empty = true;
    private AudioFormat format;
    private AudioFloatConverter converter;
    private byte[] converter_buffer;

    public SoftAudioBuffer(int size, AudioFormat format) {
        this.size = size;
        this.format = format;
        converter = AudioFloatConverter.getConverter(format);
    }

    public void swap(SoftAudioBuffer swap)
    {
        int bak_size = size;
        float[] bak_buffer = buffer;
        boolean bak_empty = empty;
        AudioFormat bak_format = format;
        AudioFloatConverter bak_converter = converter;
        byte[] bak_converter_buffer = converter_buffer;

        size = swap.size;
```

```
        buffer = swap.buffer;
        empty = swap.empty;
        format = swap.format;
        converter = swap.converter;
        converter_buffer = swap.converter_buffer;

        swap.size = bak_size;
        swap.buffer = bak_buffer;
        swap.empty = bak_empty;
        swap.format = bak_format;
        swap.converter = bak_converter;
        swap.converter_buffer = bak_converter_buffer;
    }

    public AudioFormat getFormat() {
        return format;
    }

    public int getSize() {
        return size;
    }

    public void clear() {
        if (!empty) {
            Arrays.fill(buffer, 0);
            empty = true;
        }
    }

    public boolean isSilent() {
        return empty;
    }

    public float[] array() {
        empty = false;
        if (buffer == null)
            buffer = new float[size];
        return buffer;
    }

    public void get(byte[] buffer, int channel) {

        int framesize_pc = (format.getFrameSize() / format.getChannels());
        int c_len = size * framesize_pc;
        if (converter_buffer == null || converter_buffer.length < c_len)
            converter_buffer = new byte[c_len];

        if (format.getChannels() == 1) {
            converter.toByteArray(array(), size, buffer);
        } else {
            converter.toByteArray(array(), size, converter_buffer);
            if (channel >= format.getChannels())
                return;
            int z_stepover = format.getChannels() * framesize_pc;
            int k_stepover = framesize_pc;
            for (int j = 0; j < framesize_pc; j++) {
                int k = j;
                int z = channel * framesize_pc + j;
                for (int i = 0; i < size; i++) {
                    buffer[z] = converter_buffer[k];
                    z += z_stepover;
                    k += k_stepover;
```

```
123                         }
124                     }
125                 }
126
127         }
128 }
```

79 com/sun/media/sound/SoftAudioProcessor.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * Audio processor interface.
 *
 * @author Karl Helgason
 */
public interface SoftAudioProcessor {

    public void globalParameterControlChange(int[] slothpath, long param,
            long value);

    public void init(float samplerate, float controlrate);

    public void setInput(int pin, SoftAudioBuffer input);

    public void setOutput(int pin, SoftAudioBuffer output);

    public void setMixMode(boolean mix);

    public void processAudio();

    public void processControlLogic();
}
```

80 com/sun/media/sound/SoftAudioPusher.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.io.IOException;

import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.SourceDataLine;

/**
 * This is a processor object that writes into SourceDataLine
 *
 * @author Karl Helgason
 */
public class SoftAudioPusher implements Runnable {

    private volatile boolean active = false;
    private SourceDataLine sourceDataLine = null;
    private Thread audiothread;
    private AudioInputStream ais;
    private byte[] buffer;

    public SoftAudioPusher(SourceDataLine sourceDataLine, AudioInputStream ais,
            int workbuffersizer) {
        this.ais = ais;
        this.buffer = new byte[workbuffersizer];
        this.sourceDataLine = sourceDataLine;
    }

    public synchronized void start() {
        if (active)
            return;
        active = true;
        audiothread = new Thread(this);
        audiothread.setDaemon(true);
        audiothread.setPriority(Thread.MAX_PRIORITY);
        audiothread.start();
    }
```

```
61
62      public synchronized void stop() {
63          if (!active)
64              return;
65          active = false;
66          try {
67              audiothread.join();
68          } catch (InterruptedException e) {
69              //e.printStackTrace();
70          }
71      }
72
73      public void run() {
74          byte[] buffer = SoftAudioPusher.this.buffer;
75          AudioInputStream ais = SoftAudioPusher.this.ais;
76          SourceDataLine sourceDataLine = SoftAudioPusher.this.sourceDataLine;
77
78          try {
79              while (active) {
80                  // Read from audio source
81                  int count = ais.read(buffer);
82                  if(count < 0) break;
83                  // Write byte buffer to source output
84                  sourceDataLine.write(buffer, 0, count);
85              }
86          } catch (IOException e) {
87              active = false;
88              //e.printStackTrace();
89          }
90
91      }
92  }
```

81 com/sun/media/sound/SoftChannel.java

```
/*
 * Copyright 2007-2010 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

import javax.sound.midi.MidiChannel;
import javax.sound.midi.Patch;

/**
 * Software Synthesizer MIDI channel class.
 *
 * @author Karl Helgason
 */
public class SoftChannel implements MidiChannel, ModelDirectedPlayer {

    private static boolean[] dontResetControls = new boolean[128];
    static {
        for (int i = 0; i < dontResetControls.length; i++)
            dontResetControls[i] = false;

        dontResetControls[0] = true;   // Bank Select (MSB)
        dontResetControls[32] = true;  // Bank Select (LSB)
        dontResetControls[7] = true;   // Channel Volume (MSB)
        dontResetControls[8] = true;   // Balance (MSB)
        dontResetControls[10] = true;  // Pan (MSB)
        dontResetControls[11] = true;  // Expression (MSB)
        dontResetControls[91] = true;  // Effects 1 Depth (default: Reverb Send)
        dontResetControls[92] = true;  // Effects 2 Depth (default: Tremolo Depth)
        dontResetControls[93] = true;  // Effects 3 Depth (default: Chorus Send)
        dontResetControls[94] = true;  // Effects 4 Depth (default: Celeste [Detune] Depth)
        dontResetControls[95] = true;  // Effects 5 Depth (default: Phaser Depth)
        dontResetControls[70] = true;  // Sound Controller 1 (default: Sound Variation)
```

```
        dontResetControls[71] = true;  // Sound Controller 2 (default: Timbre / Harmonic Quality)
        dontResetControls[72] = true;  // Sound Controller 3 (default: Release Time)
        dontResetControls[73] = true;  // Sound Controller 4 (default: Attack Time)
        dontResetControls[74] = true;  // Sound Controller 5 (default: Brightness)
        dontResetControls[75] = true;  // Sound Controller 6 (GM2 default: Decay Time)
        dontResetControls[76] = true;  // Sound Controller 7 (GM2 default: Vibrato Rate)
        dontResetControls[77] = true;  // Sound Controller 8 (GM2 default: Vibrato Depth)
        dontResetControls[78] = true;  // Sound Controller 9 (GM2 default: Vibrato Delay)
        dontResetControls[79] = true;  // Sound Controller 10 (GM2 default: Undefined)
        dontResetControls[120] = true; // All Sound Off
        dontResetControls[121] = true; // Reset All Controllers
        dontResetControls[122] = true; // Local Control On/Off
        dontResetControls[123] = true; // All Notes Off
        dontResetControls[124] = true; // Omni Mode Off
        dontResetControls[125] = true; // Omni Mode On
        dontResetControls[126] = true; // Poly Mode Off
        dontResetControls[127] = true; // Poly Mode On

        dontResetControls[6] = true;   // Data Entry (MSB)
        dontResetControls[38] = true;  // Data Entry (LSB)
        dontResetControls[96] = true;  // Data Increment
        dontResetControls[97] = true;  // Data Decrement
        dontResetControls[98] = true;  // Non-Registered Parameter Number (LSB)
        dontResetControls[99] = true;  // Non-Registered Parameter Number(MSB)
        dontResetControls[100] = true; // RPN = Null
        dontResetControls[101] = true; // RPN = Null

    }

    private static final int RPN_NULL_VALUE = (127 << 7) + 127;
    private int rpn_control = RPN_NULL_VALUE;
    private int nrpn_control = RPN_NULL_VALUE;
    protected double portamento_time = 1; // keyschanges per control buffer time
    protected int[] portamento_lastnote = new int[128];
    protected int portamento_lastnote_ix = 0;
    private boolean portamento = false;
    private boolean mono = false;
    private boolean mute = false;
    private boolean solo = false;
    private boolean solomute = false;
    private Object control_mutex;
    private int channel;
    private SoftVoice[] voices;
    private int bank;
    private int program;
    private SoftSynthesizer synthesizer;
    private SoftMainMixer mainmixer;
    private int[] polypressure = new int[128];
    private int channelpressure = 0;
    private int[] controller = new int[128];
    private int pitchbend;
    private double[] co_midi_pitch = new double[1];
    private double[] co_midi_channel_pressure = new double[1];
    protected SoftTuning tuning = new SoftTuning();
    protected int tuning_bank = 0;
    protected int tuning_program = 0;
    protected SoftInstrument current_instrument = null;
    protected ModelChannelMixer current_mixer = null;
    protected ModelDirector current_director = null;

    // Controller Destination Settings
    protected int cds_control_number = -1;
```

```
    protected ModelConnectionBlock[] cds_control_connections = null;
    protected ModelConnectionBlock[] cds_channelpressure_connections = null;
    protected ModelConnectionBlock[] cds_polypressure_connections = null;
    protected boolean sustain = false;
    protected boolean[][] keybasedcontroller_active = null;
    protected double[][] keybasedcontroller_value = null;

    private class MidiControlObject implements SoftControl {
        double[] pitch = co_midi_pitch;
        double[] channel_pressure = co_midi_channel_pressure;
        double[] poly_pressure = new double[1];

        public double[] get(int instance, String name) {
            if (name == null)
                return null;
            if (name.equals("pitch"))
                return pitch;
            if (name.equals("channel_pressure"))
                return channel_pressure;
            if (name.equals("poly_pressure"))
                return poly_pressure;
            return null;
        }
    }

    private SoftControl[] co_midi = new SoftControl[128];
    {
        for (int i = 0; i < co_midi.length; i++) {
            co_midi[i] = new MidiControlObject();
        }
    }

    private double[][] co_midi_cc_cc = new double[128][1];
    private SoftControl co_midi_cc = new SoftControl() {
        double[][] cc = co_midi_cc_cc;
        public double[] get(int instance, String name) {
            if (name == null)
                return null;
            return cc[Integer.parseInt(name)];
        }
    };
    Map<Integer, int[]> co_midi_rpn_rpn_i = new HashMap<Integer, int[]>();
    Map<Integer, double[]> co_midi_rpn_rpn = new HashMap<Integer, double[]>();
    private SoftControl co_midi_rpn = new SoftControl() {
        Map<Integer, double[]> rpn = co_midi_rpn_rpn;
        public double[] get(int instance, String name) {
            if (name == null)
                return null;
            int iname = Integer.parseInt(name);
            double[] v = rpn.get(iname);
            if (v == null) {
                v = new double[1];
                rpn.put(iname, v);
            }
            return v;
        }
    };
    Map<Integer, int[]> co_midi_nrpn_nrpn_i = new HashMap<Integer, int[]>();
    Map<Integer, double[]> co_midi_nrpn_nrpn = new HashMap<Integer, double[]>();
    private SoftControl co_midi_nrpn = new SoftControl() {
        Map<Integer, double[]> nrpn = co_midi_nrpn_nrpn;
        public double[] get(int instance, String name) {
```

```
            if (name == null)
                return null;
            int iname = Integer.parseInt(name);
            double[] v = nrpn.get(iname);
            if (v == null) {
                v = new double[1];
                nrpn.put(iname, v);
            }
            return v;
        }
    };

    private static int restrict7Bit(int value)
    {
        if(value < 0) return 0;
        if(value > 127) return 127;
        return value;
    }

    private static int restrict14Bit(int value)
    {
        if(value < 0) return 0;
        if(value > 16256) return 16256;
        return value;
    }

    public SoftChannel(SoftSynthesizer synth, int channel) {
        this.channel = channel;
        this.voices = synth.getVoices();
        this.synthesizer = synth;
        this.mainmixer = synth.getMainMixer();
        control_mutex = synth.control_mutex;
        resetAllControllers(true);
    }

    private int findFreeVoice(int x) {
        if(x == -1)
        {
            // x = -1 means that there where no available voice
            // last time we called findFreeVoice
            // and it hasn't changed because no audio has been
            // rendered in the meantime.
            // Therefore we have to return -1.
            return -1;
        }
        for (int i = x; i < voices.length; i++)
            if (!voices[i].active)
                return i;

        // No free voice was found, we must steal one

        int vmode = synthesizer.getVoiceAllocationMode();
        if (vmode == 1) {
            // DLS Static Voice Allocation

            //  * priority ( 10, 1-9, 11-16)
            // Search for channel to steal from
            int steal_channel = channel;
            for (int j = 0; j < voices.length; j++) {
                if (voices[j].stealer_channel == null) {
                    if (steal_channel == 9) {
                        steal_channel = voices[j].channel;
```

```
                } else {
                    if (voices[j].channel != 9) {
                        if (voices[j].channel > steal_channel)
                            steal_channel = voices[j].channel;
                    }
                }
            }
        }

        int voiceNo = -1;

        SoftVoice v = null;
        // Search for oldest voice in off state on steal_channel
        for (int j = 0; j < voices.length; j++) {
            if (voices[j].channel == steal_channel) {
                if (voices[j].stealer_channel == null && !voices[j].on) {
                    if (v == null) {
                        v = voices[j];
                        voiceNo = j;
                    }
                    if (voices[j].voiceID < v.voiceID) {
                        v = voices[j];
                        voiceNo = j;
                    }
                }
            }
        }
        // Search for oldest voice in on state on steal_channel
        if (voiceNo == -1) {
            for (int j = 0; j < voices.length; j++) {
                if (voices[j].channel == steal_channel) {
                    if (voices[j].stealer_channel == null) {
                        if (v == null) {
                            v = voices[j];
                            voiceNo = j;
                        }
                        if (voices[j].voiceID < v.voiceID) {
                            v = voices[j];
                            voiceNo = j;
                        }
                    }
                }
            }
        }

        return voiceNo;

    } else {
        // Default Voice Allocation
        //  * Find voice that is on
        //      and Find voice which has lowest voiceID ( oldest voice)
        //  * Or find voice that is off
        //      and Find voice which has lowest voiceID ( oldest voice)

        int voiceNo = -1;

        SoftVoice v = null;
        // Search for oldest voice in off state
        for (int j = 0; j < voices.length; j++) {
            if (voices[j].stealer_channel == null && !voices[j].on) {
                if (v == null) {
                    v = voices[j];
```

```
                        voiceNo = j;
                    }
                    if (voices[j].voiceID < v.voiceID) {
                        v = voices[j];
                        voiceNo = j;
                    }
                }
            }
            // Search for oldest voice in on state
            if (voiceNo == -1) {

                for (int j = 0; j < voices.length; j++) {
                    if (voices[j].stealer_channel == null) {
                        if (v == null) {
                            v = voices[j];
                            voiceNo = j;
                        }
                        if (voices[j].voiceID < v.voiceID) {
                            v = voices[j];
                            voiceNo = j;
                        }
                    }
                }
            }

            return voiceNo;
        }

    }

    protected void initVoice(SoftVoice voice, SoftPerformer p, int voiceID,
            int noteNumber, int velocity, int delay, ModelConnectionBlock[] connectionBlocks,
            ModelChannelMixer channelmixer, boolean releaseTriggered) {
        if (voice.active) {
            // Voice is active , we must steal the voice
            voice.stealer_channel = this;
            voice.stealer_performer = p;
            voice.stealer_voiceID = voiceID;
            voice.stealer_noteNumber = noteNumber;
            voice.stealer_velocity = velocity;
            voice.stealer_extendedConnectionBlocks = connectionBlocks;
            voice.stealer_channelmixer = channelmixer;
            voice.stealer_releaseTriggered = releaseTriggered;
            for (int i = 0; i < voices.length; i++)
                if (voices[i].active && voices[i].voiceID == voice.voiceID)
                    voices[i].soundOff();
            return;
        }

        voice.extendedConnectionBlocks = connectionBlocks;
        voice.channelmixer = channelmixer;
        voice.releaseTriggered = releaseTriggered;
        voice.voiceID = voiceID;
        voice.tuning = tuning;
        voice.exclusiveClass = p.exclusiveClass;
        voice.softchannel = this;
        voice.channel = channel;
        voice.bank = bank;
        voice.program = program;
        voice.instrument = current_instrument;
        voice.performer = p;
        voice.objects.clear();
```

```
371         voice.objects.put("midi", co_midi[noteNumber]);
372         voice.objects.put("midi_cc", co_midi_cc);
373         voice.objects.put("midi_rpn", co_midi_rpn);
374         voice.objects.put("midi_nrpn", co_midi_nrpn);
375         voice.noteOn(noteNumber, velocity, delay);
376         voice.setMute(mute);
377         voice.setSoloMute(solomute);
378         if (releaseTriggered)
379             return;
380         if (controller[84] != 0) {
381             voice.co_noteon_keynumber[0]
382                     = (tuning.getTuning(controller[84]) / 100.0)
383                     * (1f / 128f);
384             voice.portamento = true;
385             controlChange(84, 0);
386         } else if (portamento) {
387             if (mono) {
388                 if (portamento_lastnote[0] != -1) {
389                     voice.co_noteon_keynumber[0]
390                             = (tuning.getTuning(portamento_lastnote[0]) / 100.0)
391                             * (1f / 128f);
392                     voice.portamento = true;
393                     controlChange(84, 0);
394                 }
395                 portamento_lastnote[0] = noteNumber;
396             } else {
397                 if (portamento_lastnote_ix != 0) {
398                     portamento_lastnote_ix--;
399                     voice.co_noteon_keynumber[0]
400                             = (tuning.getTuning(
401                                     portamento_lastnote[portamento_lastnote_ix])
402                                 / 100.0)
403                             * (1f / 128f);
404                     voice.portamento = true;
405                 }
406             }
407         }
408     }
409 
410     public void noteOn(int noteNumber, int velocity) {
411         noteOn(noteNumber, velocity, 0);
412     }
413 
414     /* A special noteOn with delay parameter, which is used to
415      * start note within control buffers.
416      */
417     protected void noteOn(int noteNumber, int velocity, int delay) {
418         noteNumber = restrict7Bit(noteNumber);
419         velocity = restrict7Bit(velocity);
420         noteOn_internal(noteNumber, velocity, delay);
421         if (current_mixer != null)
422             current_mixer.noteOn(noteNumber, velocity);
423     }
424 
425     private void noteOn_internal(int noteNumber, int velocity, int delay) {
426 
427         if (velocity == 0) {
428             noteOff_internal(noteNumber, 64);
429             return;
430         }
431 
432         synchronized (control_mutex) {
```

```
433             if (sustain) {
434                 sustain = false;
435                 for (int i = 0; i < voices.length; i++) {
436                     if ((voices[i].sustain || voices[i].on)
437                             && voices[i].channel == channel && voices[i].active
438                             && voices[i].note == noteNumber) {
439                         voices[i].sustain = false;
440                         voices[i].on = true;
441                         voices[i].noteOff(0);
442                     }
443                 }
444                 sustain = true;
445             }
446 
447             mainmixer.activity();
448 
449             if (mono) {
450                 if (portamento) {
451                     boolean n_found = false;
452                     for (int i = 0; i < voices.length; i++) {
453                         if (voices[i].on && voices[i].channel == channel
454                                 && voices[i].active
455                                 && voices[i].releaseTriggered == false) {
456                             voices[i].portamento = true;
457                             voices[i].setNote(noteNumber);
458                             n_found = true;
459                         }
460                     }
461                     if (n_found) {
462                         portamento_lastnote[0] = noteNumber;
463                         return;
464                     }
465                 }
466 
467                 if (controller[84] != 0) {
468                     boolean n_found = false;
469                     for (int i = 0; i < voices.length; i++) {
470                         if (voices[i].on && voices[i].channel == channel
471                                 && voices[i].active
472                                 && voices[i].note == controller[84]
473                                 && voices[i].releaseTriggered == false) {
474                             voices[i].portamento = true;
475                             voices[i].setNote(noteNumber);
476                             n_found = true;
477                         }
478                     }
479                     controlChange(84, 0);
480                     if (n_found)
481                         return;
482                 }
483             }
484 
485             if (mono)
486                 allNotesOff();
487 
488             if (current_instrument == null) {
489                 current_instrument
490                         = synthesizer.findInstrument(program, bank, channel);
491                 if (current_instrument == null)
492                     return;
493                 if (current_mixer != null)
494                     mainmixer.stopMixer(current_mixer);
```

```
                current_mixer = current_instrument.getSourceInstrument()
                        .getChannelMixer(this, synthesizer.getFormat());
                if (current_mixer != null)
                    mainmixer.registerMixer(current_mixer);
                current_director = current_instrument.getDirector(this, this);
                applyInstrumentCustomization();
            }
            prevVoiceID = synthesizer.voiceIDCounter++;
            firstVoice = true;
            voiceNo = 0;

            int tunedKey = (int)(Math.round(tuning.getTuning()[noteNumber]/100.0));
            play_noteNumber = noteNumber;
            play_velocity = velocity;
            play_delay = delay;
            play_releasetriggered = false;
            lastVelocity[noteNumber] = velocity;
            current_director.noteOn(tunedKey, velocity);

            /*
            SoftPerformer[] performers = current_instrument.getPerformers();
            for (int i = 0; i < performers.length; i++) {
                SoftPerformer p = performers[i];
                if (p.keyFrom <= tunedKey && p.keyTo >= tunedKey) {
                    if (p.velFrom <= velocity && p.velTo >= velocity) {
                        if (firstVoice) {
                            firstVoice = false;
                            if (p.exclusiveClass != 0) {
                                int x = p.exclusiveClass;
                                for (int j = 0; j < voices.length; j++) {
                                    if (voices[j].active
                                            && voices[j].channel == channel
                                            && voices[j].exclusiveClass == x) {
                                        if (!(p.selfNonExclusive
                                                && voices[j].note == noteNumber))
                                            voices[j].shutdown();
                                    }
                                }
                            }
                        }
                        voiceNo = findFreeVoice(voiceNo);
                        if (voiceNo == -1)
                            return;
                        initVoice(voices[voiceNo], p, prevVoiceID, noteNumber,
                                velocity);
                    }
                }
            }
            */
        }
    }

    public void noteOff(int noteNumber, int velocity) {
        noteNumber = restrict7Bit(noteNumber);
        velocity = restrict7Bit(velocity);
        noteOff_internal(noteNumber, velocity);

        if (current_mixer != null)
            current_mixer.noteOff(noteNumber, velocity);
    }

    private void noteOff_internal(int noteNumber, int velocity) {
```

```
        synchronized (control_mutex) {

            if (!mono) {
                if (portamento) {
                    if (portamento_lastnote_ix != 127) {
                        portamento_lastnote[portamento_lastnote_ix] = noteNumber;
                        portamento_lastnote_ix++;
                    }
                }
            }

            mainmixer.activity();
            for (int i = 0; i < voices.length; i++) {
                if (voices[i].on && voices[i].channel == channel
                        && voices[i].note == noteNumber
                        && voices[i].releaseTriggered == false) {
                    voices[i].noteOff(velocity);
                }
                // We must also check stolen voices
                if (voices[i].stealer_channel == this && voices[i].stealer_noteNumber ==
                    noteNumber) {
                    SoftVoice v = voices[i];
                    v.stealer_releaseTriggered = false;
                    v.stealer_channel = null;
                    v.stealer_performer = null;
                    v.stealer_voiceID = -1;
                    v.stealer_noteNumber = 0;
                    v.stealer_velocity = 0;
                    v.stealer_extendedConnectionBlocks = null;
                    v.stealer_channelmixer = null;
                }
            }

            // Try play back note-off triggered voices,

            if (current_instrument == null) {
                current_instrument
                        = synthesizer.findInstrument(program, bank, channel);
                if (current_instrument == null)
                    return;
                if (current_mixer != null)
                    mainmixer.stopMixer(current_mixer);
                current_mixer = current_instrument.getSourceInstrument()
                        .getChannelMixer(this, synthesizer.getFormat());
                if (current_mixer != null)
                    mainmixer.registerMixer(current_mixer);
                current_director = current_instrument.getDirector(this, this);
                applyInstrumentCustomization();

            }
            prevVoiceID = synthesizer.voiceIDCounter++;
            firstVoice = true;
            voiceNo = 0;

            int tunedKey = (int)(Math.round(tuning.getTuning()[noteNumber]/100.0));
            play_noteNumber = noteNumber;
            play_velocity = lastVelocity[noteNumber];
            play_releasetriggered = true;
            play_delay = 0;
            current_director.noteOff(tunedKey, velocity);

        }
```

```
    }
    private int[] lastVelocity = new int[128];
    private int prevVoiceID;
    private boolean firstVoice = true;
    private int voiceNo = 0;
    private int play_noteNumber = 0;
    private int play_velocity = 0;
    private int play_delay = 0;
    private boolean play_releasetriggered = false;

    public void play(int performerIndex, ModelConnectionBlock[] connectionBlocks) {

        int noteNumber = play_noteNumber;
        int velocity = play_velocity;
        int delay = play_delay;
        boolean releasetriggered = play_releasetriggered;

        SoftPerformer p = current_instrument.getPerformers()[performerIndex];

        if (firstVoice) {
            firstVoice = false;
            if (p.exclusiveClass != 0) {
                int x = p.exclusiveClass;
                for (int j = 0; j < voices.length; j++) {
                    if (voices[j].active && voices[j].channel == channel
                            && voices[j].exclusiveClass == x) {
                        if (!(p.selfNonExclusive && voices[j].note == noteNumber))
                            voices[j].shutdown();
                    }
                }
            }
        }

        voiceNo = findFreeVoice(voiceNo);

        if (voiceNo == -1)
            return;

        initVoice(voices[voiceNo], p, prevVoiceID, noteNumber, velocity, delay,
                connectionBlocks, current_mixer, releasetriggered);
    }

    public void noteOff(int noteNumber) {
        if(noteNumber < 0 || noteNumber > 127) return;
        noteOff_internal(noteNumber, 64);
    }

    public void setPolyPressure(int noteNumber, int pressure) {
        noteNumber = restrict7Bit(noteNumber);
        pressure = restrict7Bit(pressure);

        if (current_mixer != null)
            current_mixer.setPolyPressure(noteNumber, pressure);

        synchronized (control_mutex) {
            mainmixer.activity();
            co_midi[noteNumber].get(0, "poly_pressure")[0] = pressure*(1.0/128.0);
            polypressure[noteNumber] = pressure;
            for (int i = 0; i < voices.length; i++) {
                if (voices[i].active && voices[i].note == noteNumber)
                    voices[i].setPolyPressure(pressure);
            }
```

```
        }
    }

    public int getPolyPressure(int noteNumber) {
        synchronized (control_mutex) {
            return polypressure[noteNumber];
        }
    }

    public void setChannelPressure(int pressure) {
        pressure = restrict7Bit(pressure);
        if (current_mixer != null)
            current_mixer.setChannelPressure(pressure);
        synchronized (control_mutex) {
            mainmixer.activity();
            co_midi_channel_pressure[0] = pressure * (1.0 / 128.0);
            channelpressure = pressure;
            for (int i = 0; i < voices.length; i++) {
                if (voices[i].active)
                    voices[i].setChannelPressure(pressure);
            }
        }
    }

    public int getChannelPressure() {
        synchronized (control_mutex) {
            return channelpressure;
        }
    }

    protected void applyInstrumentCustomization() {
        if (cds_control_connections == null
                && cds_channelpressure_connections == null
                && cds_polypressure_connections == null) {
            return;
        }

        ModelInstrument src_instrument = current_instrument.getSourceInstrument();
        ModelPerformer[] performers = src_instrument.getPerformers();
        ModelPerformer[] new_performers = new ModelPerformer[performers.length];
        for (int i = 0; i < new_performers.length; i++) {
            ModelPerformer performer = performers[i];
            ModelPerformer new_performer = new ModelPerformer();
            new_performer.setName(performer.getName());
            new_performer.setExclusiveClass(performer.getExclusiveClass());
            new_performer.setKeyFrom(performer.getKeyFrom());
            new_performer.setKeyTo(performer.getKeyTo());
            new_performer.setVelFrom(performer.getVelFrom());
            new_performer.setVelTo(performer.getVelTo());
            new_performer.getOscillators().addAll(performer.getOscillators());
            new_performer.getConnectionBlocks().addAll(
                    performer.getConnectionBlocks());
            new_performers[i] = new_performer;

            List<ModelConnectionBlock> connblocks =
                    new_performer.getConnectionBlocks();

            if (cds_control_connections != null) {
                String cc = Integer.toString(cds_control_number);
                Iterator<ModelConnectionBlock> iter = connblocks.iterator();
                while (iter.hasNext()) {
                    ModelConnectionBlock conn = iter.next();
```

```
                ModelSource[] sources = conn.getSources();
                boolean removeok = false;
                if (sources != null) {
                    for (int j = 0; j < sources.length; j++) {
                        ModelSource src = sources[j];
                        if ("midi_cc".equals(src.getIdentifier().getObject())
                                && cc.equals(src.getIdentifier().getVariable())) {
                            removeok = true;
                        }
                    }
                }
                if (removeok)
                    iter.remove();
            }
            for (int j = 0; j < cds_control_connections.length; j++)
                connblocks.add(cds_control_connections[j]);
        }

        if (cds_polypressure_connections != null) {
            Iterator<ModelConnectionBlock> iter = connblocks.iterator();
            while (iter.hasNext()) {
                ModelConnectionBlock conn = iter.next();
                ModelSource[] sources = conn.getSources();
                boolean removeok = false;
                if (sources != null) {
                    for (int j = 0; j < sources.length; j++) {
                        ModelSource src = sources[j];
                        if ("midi".equals(src.getIdentifier().getObject())
                                && "poly_pressure".equals(
                                    src.getIdentifier().getVariable())) {
                            removeok = true;
                        }
                    }
                }
                if (removeok)
                    iter.remove();
            }
            for (int j = 0; j < cds_polypressure_connections.length; j++)
                connblocks.add(cds_polypressure_connections[j]);
        }


        if (cds_channelpressure_connections != null) {
            Iterator<ModelConnectionBlock> iter = connblocks.iterator();
            while (iter.hasNext()) {
                ModelConnectionBlock conn = iter.next();
                ModelSource[] sources = conn.getSources();
                boolean removeok = false;
                if (sources != null) {
                    for (int j = 0; j < sources.length; j++) {
                        ModelIdentifier srcid = sources[j].getIdentifier();
                        if ("midi".equals(srcid.getObject()) &&
                                "channel_pressure".equals(srcid.getVariable())) {
                            removeok = true;
                        }
                    }
                }
                if (removeok)
                    iter.remove();
            }
            for (int j = 0; j < cds_channelpressure_connections.length; j++)
                connblocks.add(cds_channelpressure_connections[j]);
```

```
804             }
805
806         }
807
808         current_instrument = new SoftInstrument(src_instrument, new_performers);
809
810     }
811
812     private ModelConnectionBlock[] createModelConnections(ModelIdentifier sid,
813             int[] destination, int[] range) {
814
815         /*
816         controlled parameter (pp)|range (rr)| Description             |Default
817         -------------------------|----------|-------------------------|-------
818         00 Pitch Control         | 28H..58H | -24..+24 semitones      | 40H
819         01 Filter Cutoff Control | 00H..7FH | -9600..+9450 cents      | 40H
820         02 Amplitude Control     | 00H..7FH | 0..(127/64)*100 percent | 40H
821         03 LFO Pitch Depth       | 00H..7FH | 0..600 cents            |  0
822         04 LFO Filter Depth      | 00H..7FH | 0..2400 cents           |  0
823         05 LFO Amplitude Depth   | 00H..7FH | 0..100 percent          |  0
824         */
825
826         List<ModelConnectionBlock> conns = new ArrayList<ModelConnectionBlock>();
827
828         for (int i = 0; i < destination.length; i++) {
829             int d = destination[i];
830             int r = range[i];
831             if (d == 0) {
832                 double scale = (r - 64) * 100;
833                 ModelConnectionBlock conn = new ModelConnectionBlock(
834                         new ModelSource(sid,
835                             ModelStandardTransform.DIRECTION_MIN2MAX,
836                             ModelStandardTransform.POLARITY_UNIPOLAR,
837                             ModelStandardTransform.TRANSFORM_LINEAR),
838                         scale,
839                         new ModelDestination(
840                             new ModelIdentifier("osc", "pitch")));
841                 conns.add(conn);
842
843             }
844             if (d == 1) {
845                 double scale = (r / 64.0 - 1.0) * 9600.0;
846                 ModelConnectionBlock conn;
847                 if (scale > 0) {
848                     conn = new ModelConnectionBlock(
849                             new ModelSource(sid,
850                                 ModelStandardTransform.DIRECTION_MAX2MIN,
851                                 ModelStandardTransform.POLARITY_UNIPOLAR,
852                                 ModelStandardTransform.TRANSFORM_LINEAR),
853                             -scale,
854                             new ModelDestination(
855                                 ModelDestination.DESTINATION_FILTER_FREQ));
856                 } else {
857                     conn = new ModelConnectionBlock(
858                             new ModelSource(sid,
859                                 ModelStandardTransform.DIRECTION_MIN2MAX,
860                                 ModelStandardTransform.POLARITY_UNIPOLAR,
861                                 ModelStandardTransform.TRANSFORM_LINEAR),
862                             scale,
863                             new ModelDestination(
864                                 ModelDestination.DESTINATION_FILTER_FREQ));
865                 }
```

```
                conns.add(conn);
            }
            if (d == 2) {
                final double scale = (r / 64.0);
                ModelTransform mt = new ModelTransform() {
                    double s = scale;
                    public double transform(double value) {
                        if (s < 1)
                            value = s + (value * (1.0 - s));
                        else if (s > 1)
                            value = 1 + (value * (s - 1.0));
                        else
                            return 0;
                        return -((5.0 / 12.0) / Math.log(10)) * Math.log(value);
                    }
                };

                ModelConnectionBlock conn = new ModelConnectionBlock(
                        new ModelSource(sid, mt), -960,
                        new ModelDestination(ModelDestination.DESTINATION_GAIN));
                conns.add(conn);

            }
            if (d == 3) {
                double scale = (r / 64.0 - 1.0) * 9600.0;
                ModelConnectionBlock conn = new ModelConnectionBlock(
                        new ModelSource(ModelSource.SOURCE_LFO1,
                            ModelStandardTransform.DIRECTION_MIN2MAX,
                            ModelStandardTransform.POLARITY_BIPOLAR,
                            ModelStandardTransform.TRANSFORM_LINEAR),
                        new ModelSource(sid,
                            ModelStandardTransform.DIRECTION_MIN2MAX,
                            ModelStandardTransform.POLARITY_UNIPOLAR,
                            ModelStandardTransform.TRANSFORM_LINEAR),
                        scale,
                        new ModelDestination(
                            ModelDestination.DESTINATION_PITCH));
                conns.add(conn);
            }
            if (d == 4) {
                double scale = (r / 128.0) * 2400.0;
                ModelConnectionBlock conn = new ModelConnectionBlock(
                        new ModelSource(ModelSource.SOURCE_LFO1,
                            ModelStandardTransform.DIRECTION_MIN2MAX,
                            ModelStandardTransform.POLARITY_BIPOLAR,
                            ModelStandardTransform.TRANSFORM_LINEAR),
                        new ModelSource(sid,
                            ModelStandardTransform.DIRECTION_MIN2MAX,
                            ModelStandardTransform.POLARITY_UNIPOLAR,
                            ModelStandardTransform.TRANSFORM_LINEAR),
                        scale,
                        new ModelDestination(
                            ModelDestination.DESTINATION_FILTER_FREQ));
                conns.add(conn);
            }
            if (d == 5) {
                final double scale = (r / 127.0);

                ModelTransform mt = new ModelTransform() {
                    double s = scale;
                    public double transform(double value) {
                        return -((5.0 / 12.0) / Math.log(10))
```

```
                                * Math.log(1 - value * s);
                    }
                };

                ModelConnectionBlock conn = new ModelConnectionBlock(
                        new ModelSource(ModelSource.SOURCE_LFO1,
                            ModelStandardTransform.DIRECTION_MIN2MAX,
                            ModelStandardTransform.POLARITY_UNIPOLAR,
                            ModelStandardTransform.TRANSFORM_LINEAR),
                        new ModelSource(sid, mt),
                        -960,
                        new ModelDestination(
                            ModelDestination.DESTINATION_GAIN));
                conns.add(conn);
            }
        }

        return conns.toArray(new ModelConnectionBlock[conns.size()]);
    }

    public void mapPolyPressureToDestination(int[] destination, int[] range) {
        current_instrument = null;
        if (destination.length == 0) {
            cds_polypressure_connections = null;
            return;
        }
        cds_polypressure_connections
                = createModelConnections(
                    new ModelIdentifier("midi", "poly_pressure"),
                    destination, range);
    }

    public void mapChannelPressureToDestination(int[] destination, int[] range) {
        current_instrument = null;
        if (destination.length == 0) {
            cds_channelpressure_connections = null;
            return;
        }
        cds_channelpressure_connections
                = createModelConnections(
                    new ModelIdentifier("midi", "channel_pressure"),
                    destination, range);
    }

    public void mapControlToDestination(int control, int[] destination, int[] range) {

        if (!((control >= 0x01 && control <= 0x1F)
                || (control >= 0x40 && control <= 0x5F))) {
            cds_control_connections = null;
            return;
        }

        current_instrument = null;
        cds_control_number = control;
        if (destination.length == 0) {
            cds_control_connections = null;
            return;
        }
        cds_control_connections
                = createModelConnections(
                    new ModelIdentifier("midi_cc", Integer.toString(control)),
                    destination, range);
```

```
    }

    public void controlChangePerNote(int noteNumber, int controller, int value) {

/*
 CC# | nn    | Name                      | vv              | default     | description
-----|-------|---------------------------|-----------------|-------------|---------------------------

7    |07H    |Note Volume                |00H-40H-7FH      |40H          |0-100-(127/64)*100(%)(
   Relative)
10   |0AH    |*Pan                       |00H-7FH absolute|Preset Value|Left-Center-Right (absolute)
33-63|21-3FH|LSB for                     |01H-1FH          |             |
71   |47H    |Timbre/Harmonic Intensity|00H-40H-7FH      |40H (???)    |
72   |48H    |Release Time               |00H-40H-7FH      |40H (???)    |
73   |49H    |Attack Time                |00H-40H-7FH      |40H (???)    |
74   |4AH    |Brightness                 |00H-40H-7FH      |40H (???)    |
75   |4BH    |Decay Time                 |00H-40H-7FH      |40H (???)    |
76   |4CH    |Vibrato Rate               |00H-40H-7FH      |40H (???)    |
77   |4DH    |Vibrato Depth              |00H-40H-7FH      |40H (???)    |
78   |4EH    |Vibrato Delay              |00H-40H-7FH      |40H (???)    |
91   |5BH    |*Reverb Send               |00H-7FH absolute|Preset Value|Left-Center-Right (absolute)
93   |5DH    |*Chorus Send               |00H-7FH absolute|Preset Value|Left-Center-Right (absolute)
120  |78H    |**Fine Tuning              |00H-40H-7FH      |40H (???)    |
121  |79H    |**Coarse Tuning            |00H-40H-7FH      |40H (???)    |
*/

        if (keybasedcontroller_active == null) {
            keybasedcontroller_active = new boolean[128][];
            keybasedcontroller_value = new double[128][];
        }
        if (keybasedcontroller_active[noteNumber] == null) {
            keybasedcontroller_active[noteNumber] = new boolean[128];
            Arrays.fill(keybasedcontroller_active[noteNumber], false);
            keybasedcontroller_value[noteNumber] = new double[128];
            Arrays.fill(keybasedcontroller_value[noteNumber], 0);
        }

        if (value == -1) {
            keybasedcontroller_active[noteNumber][controller] = false;
        } else {
            keybasedcontroller_active[noteNumber][controller] = true;
            keybasedcontroller_value[noteNumber][controller] = value / 128.0;
        }

        if (controller < 120) {
            for (int i = 0; i < voices.length; i++)
                if (voices[i].active)
                    voices[i].controlChange(controller, -1);
        } else if (controller == 120) {
            for (int i = 0; i < voices.length; i++)
                if (voices[i].active)
                    voices[i].rpnChange(1, -1);
        } else if (controller == 121) {
            for (int i = 0; i < voices.length; i++)
                if (voices[i].active)
                    voices[i].rpnChange(2, -1);
        }

    }

    public int getControlPerNote(int noteNumber, int controller) {
        if (keybasedcontroller_active == null)
```

```
            return -1;
        if (keybasedcontroller_active[noteNumber] == null)
            return -1;
        if (!keybasedcontroller_active[noteNumber][controller])
            return -1;
        return (int)(keybasedcontroller_value[noteNumber][controller] * 128);
    }

    public void controlChange(int controller, int value) {
        controller = restrict7Bit(controller);
        value = restrict7Bit(value);
        if (current_mixer != null)
            current_mixer.controlChange(controller, value);

        synchronized (control_mutex) {
            switch (controller) {
            /*
            Map<String, int[]>co_midi_rpn_rpn_i = new HashMap<String, int[]>();
            Map<String, double[]>co_midi_rpn_rpn = new HashMap<String, double[]>();
            Map<String, int[]>co_midi_nrpn_nrpn_i = new HashMap<String, int[]>();
            Map<String, double[]>co_midi_nrpn_nrpn = new HashMap<String, double[]>();
             */

            case 5:
                // This produce asin-like curve
                // as described in General Midi Level 2 Specification, page 6
                double x = -Math.asin((value / 128.0) * 2 - 1) / Math.PI + 0.5;
                x = Math.pow(100000.0, x) / 100.0;  // x is now cent/msec
                // Convert x from cent/msec to key/controlbuffertime
                x = x / 100.0;                      // x is now keys/msec
                x = x * 1000.0;                     // x is now keys/sec
                x = x / synthesizer.getControlRate(); // x is now keys/controlbuffertime
                portamento_time = x;
                break;
            case 6:
            case 38:
            case 96:
            case 97:
                int val = 0;
                if (nrpn_control != RPN_NULL_VALUE) {
                    int[] val_i = co_midi_nrpn_nrpn_i.get(nrpn_control);
                    if (val_i != null)
                        val = val_i[0];
                }
                if (rpn_control != RPN_NULL_VALUE) {
                    int[] val_i = co_midi_rpn_rpn_i.get(rpn_control);
                    if (val_i != null)
                        val = val_i[0];
                }

                if (controller == 6)
                    val = (val & 127) + (value << 7);
                else if (controller == 38)
                    val = (val & (127 << 7)) + value;
                else if (controller == 96 || controller == 97) {
                    int step = 1;
                    if (rpn_control == 2 || rpn_control == 3 || rpn_control == 4)
                        step = 128;
                    if (controller == 96)
                        val += step;
                    if (controller == 97)
                        val -= step;
```

```
            }

            if (nrpn_control != RPN_NULL_VALUE)
                nrpnChange(nrpn_control, val);
            if (rpn_control != RPN_NULL_VALUE)
                rpnChange(rpn_control, val);

            break;
        case 64: // Hold1 (Damper) (cc#64)
            boolean on = value >= 64;
            if (sustain != on) {
                sustain = on;
                if (!on) {
                    for (int i = 0; i < voices.length; i++) {
                        if (voices[i].active && voices[i].sustain &&
                                voices[i].channel == channel) {
                            voices[i].sustain = false;
                            if (!voices[i].on) {
                                voices[i].on = true;
                                voices[i].noteOff(0);
                            }
                        }
                    }
                } else {
                    for (int i = 0; i < voices.length; i++)
                        if (voices[i].active && voices[i].channel == channel)
                            voices[i].redamp();
                }
            }
            break;
        case 65:
            //allNotesOff();
            portamento = value >= 64;
            portamento_lastnote[0] = -1;
            /*
            for (int i = 0; i < portamento_lastnote.length; i++)
                portamento_lastnote[i] = -1;
             */
            portamento_lastnote_ix = 0;
            break;
        case 66: // Sostenuto (cc#66)
            on = value >= 64;
            if (on) {
                for (int i = 0; i < voices.length; i++) {
                    if (voices[i].active && voices[i].on &&
                            voices[i].channel == channel) {
                        voices[i].sostenuto = true;
                    }
                }
            }
            if (!on) {
                for (int i = 0; i < voices.length; i++) {
                    if (voices[i].active && voices[i].sostenuto &&
                            voices[i].channel == channel) {
                        voices[i].sostenuto = false;
                        if (!voices[i].on) {
                            voices[i].on = true;
                            voices[i].noteOff(0);
                        }
                    }
                }
            }
```

```
            break;
        case 98:
            nrpn_control = (nrpn_control & (127 << 7)) + value;
            rpn_control = RPN_NULL_VALUE;
            break;
        case 99:
            nrpn_control = (nrpn_control & 127) + (value << 7);
            rpn_control = RPN_NULL_VALUE;
            break;
        case 100:
            rpn_control = (rpn_control & (127 << 7)) + value;
            nrpn_control = RPN_NULL_VALUE;
            break;
        case 101:
            rpn_control = (rpn_control & 127) + (value << 7);
            nrpn_control = RPN_NULL_VALUE;
            break;
        case 120:
            allSoundOff();
            break;
        case 121:
            resetAllControllers(value == 127);
            break;
        case 122:
            localControl(value >= 64);
            break;
        case 123:
            allNotesOff();
            break;
        case 124:
            setOmni(false);
            break;
        case 125:
            setOmni(true);
            break;
        case 126:
            if (value == 1)
                setMono(true);
            break;
        case 127:
            setMono(false);
            break;

        default:
            break;
        }

        co_midi_cc_cc[controller][0] = value * (1.0 / 128.0);

        if (controller == 0x00) {
            bank = /*(bank & 127) +*/ (value << 7);
            return;
        }

        if (controller == 0x20) {
            bank = (bank & (127 << 7)) + value;
            return;
        }

        this.controller[controller] = value;
        if(controller < 0x20)
            this.controller[controller + 0x20] = 0;
```

```
            for (int i = 0; i < voices.length; i++)
                if (voices[i].active)
                    voices[i].controlChange(controller, value);

        }
    }

    public int getController(int controller) {
        synchronized (control_mutex) {
            // Should only return lower 7 bits,
            // even when controller is "boosted" higher.
            return this.controller[controller] & 127;
        }
    }

    public void tuningChange(int program) {
        tuningChange(0, program);
    }

    public void tuningChange(int bank, int program) {
        synchronized (control_mutex) {
            tuning = synthesizer.getTuning(new Patch(bank, program));
        }
    }

    public void programChange(int program) {
        programChange(bank, program);
    }

    public void programChange(int bank, int program) {
        bank = restrict14Bit(bank);
        program = restrict7Bit(program);
        synchronized (control_mutex) {
            mainmixer.activity();
            if(this.bank != bank || this.program != program)
            {
                this.bank = bank;
                this.program = program;
                current_instrument = null;
            }
        }
    }

    public int getProgram() {
        synchronized (control_mutex) {
            return program;
        }
    }

    public void setPitchBend(int bend) {
        bend = restrict14Bit(bend);
        if (current_mixer != null)
            current_mixer.setPitchBend(bend);
        synchronized (control_mutex) {
            mainmixer.activity();
            co_midi_pitch[0] = bend * (1.0 / 16384.0);
            pitchbend = bend;
            for (int i = 0; i < voices.length; i++)
                if (voices[i].active)
                    voices[i].setPitchBend(bend);
        }
```

```
    }

    public int getPitchBend() {
        synchronized (control_mutex) {
            return pitchbend;
        }
    }

    public void nrpnChange(int controller, int value) {

        /*
        System.out.println("(" + channel + ").nrpnChange("
                + Integer.toHexString(controller >> 7)
                + " " + Integer.toHexString(controller & 127)
                + ", " + Integer.toHexString(value >> 7)
                + " " + Integer.toHexString(value & 127) + ")");
         */

        if (synthesizer.getGeneralMidiMode() == 0) {
            if (controller == (0x01 << 7) + (0x08)) // Vibrato Rate
                controlChange(76, value >> 7);
            if (controller == (0x01 << 7) + (0x09)) // Vibrato Depth
                controlChange(77, value >> 7);
            if (controller == (0x01 << 7) + (0x0A)) // Vibrato Delay
                controlChange(78, value >> 7);
            if (controller == (0x01 << 7) + (0x20)) // Brightness
                controlChange(74, value >> 7);
            if (controller == (0x01 << 7) + (0x21)) // Filter Resonance
                controlChange(71, value >> 7);
            if (controller == (0x01 << 7) + (0x63)) // Attack Time
                controlChange(73, value >> 7);
            if (controller == (0x01 << 7) + (0x64)) // Decay Time
                controlChange(75, value >> 7);
            if (controller == (0x01 << 7) + (0x66)) // Release Time
                controlChange(72, value >> 7);

            if (controller >> 7 == 0x18) // Pitch coarse
                controlChangePerNote(controller % 128, 120, value >> 7);
            if (controller >> 7 == 0x1A) // Volume
                controlChangePerNote(controller % 128, 7, value >> 7);
            if (controller >> 7 == 0x1C) // Panpot
                controlChangePerNote(controller % 128, 10, value >> 7);
            if (controller >> 7 == 0x1D) // Reverb
                controlChangePerNote(controller % 128, 91, value >> 7);
            if (controller >> 7 == 0x1E) // Chorus
                controlChangePerNote(controller % 128, 93, value >> 7);
        }

        int[] val_i = co_midi_nrpn_nrpn_i.get(controller);
        double[] val_d = co_midi_nrpn_nrpn.get(controller);
        if (val_i == null) {
            val_i = new int[1];
            co_midi_nrpn_nrpn_i.put(controller, val_i);
        }
        if (val_d == null) {
            val_d = new double[1];
            co_midi_nrpn_nrpn.put(controller, val_d);
        }
        val_i[0] = value;
        val_d[0] = val_i[0] * (1.0 / 16384.0);

        for (int i = 0; i < voices.length; i++)
```

```
            if (voices[i].active)
                voices[i].nrpnChange(controller, val_i[0]);

    }

    public void rpnChange(int controller, int value) {

        /*
        System.out.println("(" + channel + ").rpnChange("
                + Integer.toHexString(controller >> 7)
                + " " + Integer.toHexString(controller & 127)
                + ", " + Integer.toHexString(value >> 7)
                + " " + Integer.toHexString(value & 127) + ")");
         */

        if (controller == 3) {
            tuning_program = (value >> 7) & 127;
            tuningChange(tuning_bank, tuning_program);
        }
        if (controller == 4) {
            tuning_bank = (value >> 7) & 127;
        }

        int[] val_i = co_midi_rpn_rpn_i.get(controller);
        double[] val_d = co_midi_rpn_rpn.get(controller);
        if (val_i == null) {
            val_i = new int[1];
            co_midi_rpn_rpn_i.put(controller, val_i);
        }
        if (val_d == null) {
            val_d = new double[1];
            co_midi_rpn_rpn.put(controller, val_d);
        }
        val_i[0] = value;
        val_d[0] = val_i[0] * (1.0 / 16384.0);

        for (int i = 0; i < voices.length; i++)
            if (voices[i].active)
                voices[i].rpnChange(controller, val_i[0]);
    }

    public void resetAllControllers() {
        resetAllControllers(false);
    }

    public void resetAllControllers(boolean allControls) {
        synchronized (control_mutex) {
            mainmixer.activity();

            for (int i = 0; i < 128; i++) {
                setPolyPressure(i, 0);
            }
            setChannelPressure(0);
            setPitchBend(8192);
            for (int i = 0; i < 128; i++) {
                if (!dontResetControls[i])
                    controlChange(i, 0);
            }

            controlChange(71, 64); // Filter Resonance
            controlChange(72, 64); // Release Time
            controlChange(73, 64); // Attack Time
```

```
            controlChange(74, 64); // Brightness
            controlChange(75, 64); // Decay Time
            controlChange(76, 64); // Vibrato Rate
            controlChange(77, 64); // Vibrato Depth
            controlChange(78, 64); // Vibrato Delay

            controlChange(8, 64); // Balance
            controlChange(11, 127); // Expression
            controlChange(98, 127); // NRPN Null
            controlChange(99, 127); // NRPN Null
            controlChange(100, 127); // RPN = Null
            controlChange(101, 127); // RPN = Null

            // see DLS 2.1 (Power-on Default Values)
            if (allControls) {

                keybasedcontroller_active = null;
                keybasedcontroller_value = null;

                controlChange(7, 100); // Volume
                controlChange(10, 64); // Pan
                controlChange(91, 40); // Reverb

                for (int controller : co_midi_rpn_rpn.keySet()) {
                    // don't reset tuning settings
                    if (controller != 3 && controller != 4)
                        rpnChange(controller, 0);
                }
                for (int controller : co_midi_nrpn_nrpn.keySet())
                    nrpnChange(controller, 0);
                rpnChange(0, 2 << 7);   // Bitch Bend sensitivity
                rpnChange(1, 64 << 7);  // Channel fine tunning
                rpnChange(2, 64 << 7);  // Channel Coarse Tuning
                rpnChange(5, 64);       // Modulation Depth, +/- 50 cent

                tuning_bank = 0;
                tuning_program = 0;
                tuning = new SoftTuning();

            }

        }
    }

    public void allNotesOff() {
        if (current_mixer != null)
            current_mixer.allNotesOff();
        synchronized (control_mutex) {
            for (int i = 0; i < voices.length; i++)
                if (voices[i].on && voices[i].channel == channel
                        && voices[i].releaseTriggered == false) {
                    voices[i].noteOff(0);
                }
        }
    }

    public void allSoundOff() {
        if (current_mixer != null)
            current_mixer.allSoundOff();
        synchronized (control_mutex) {
            for (int i = 0; i < voices.length; i++)
                if (voices[i].on && voices[i].channel == channel)
```

```
                voices[i].soundOff();
        }
    }

    public boolean localControl(boolean on) {
        return false;
    }

    public void setMono(boolean on) {
        if (current_mixer != null)
            current_mixer.setMono(on);
        synchronized (control_mutex) {
            allNotesOff();
            mono = on;
        }
    }

    public boolean getMono() {
        synchronized (control_mutex) {
            return mono;
        }
    }

    public void setOmni(boolean on) {
        if (current_mixer != null)
            current_mixer.setOmni(on);
        allNotesOff();
    // Omni is not supported by GM2
    }

    public boolean getOmni() {
        return false;
    }

    public void setMute(boolean mute) {
        if (current_mixer != null)
            current_mixer.setMute(mute);
        synchronized (control_mutex) {
            this.mute = mute;
            for (int i = 0; i < voices.length; i++)
                if (voices[i].active && voices[i].channel == channel)
                    voices[i].setMute(mute);
        }
    }

    public boolean getMute() {
        synchronized (control_mutex) {
            return mute;
        }
    }

    public void setSolo(boolean soloState) {
        if (current_mixer != null)
            current_mixer.setSolo(soloState);

        synchronized (control_mutex) {
            this.solo = soloState;

            boolean soloinuse = false;
            for (SoftChannel c : synthesizer.channels) {
                if (c.solo) {
                    soloinuse = true;
```

```
                    break;
                }
            }

            if (!soloinuse) {
                for (SoftChannel c : synthesizer.channels)
                    c.setSoloMute(false);
                return;
            }

            for (SoftChannel c : synthesizer.channels)
                c.setSoloMute(!c.solo);

        }

    }

    private void setSoloMute(boolean mute) {
        synchronized (control_mutex) {
            if (solomute == mute)
                return;
            this.solomute = mute;
            for (int i = 0; i < voices.length; i++)
                if (voices[i].active && voices[i].channel == channel)
                    voices[i].setSoloMute(solomute);
        }
    }

    public boolean getSolo() {
        synchronized (control_mutex) {
            return solo;
        }
    }
}
```

82 com/sun/media/sound/SoftChannelProxy.java

```
/*
 * Copyright 2008 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import javax.sound.midi.MidiChannel;

/**
 * A MidiChannel proxy object used for external access to synthesizer internal
 * channel objects.
 *
 * @author Karl Helgason
 */
public class SoftChannelProxy implements MidiChannel {

    private MidiChannel channel = null;

    public MidiChannel getChannel() {
        return channel;
    }

    public void setChannel(MidiChannel channel) {
        this.channel = channel;
    }

    public void allNotesOff() {
        if (channel == null)
            return;
        channel.allNotesOff();
    }

    public void allSoundOff() {
        if (channel == null)
            return;
        channel.allSoundOff();
    }

    public void controlChange(int controller, int value) {
        if (channel == null)
```

```
            return;
        channel.controlChange(controller, value);
    }

    public int getChannelPressure() {
        if (channel == null)
            return 0;
        return channel.getChannelPressure();
    }

    public int getController(int controller) {
        if (channel == null)
            return 0;
        return channel.getController(controller);
    }

    public boolean getMono() {
        if (channel == null)
            return false;
        return channel.getMono();
    }

    public boolean getMute() {
        if (channel == null)
            return false;
        return channel.getMute();
    }

    public boolean getOmni() {
        if (channel == null)
            return false;
        return channel.getOmni();
    }

    public int getPitchBend() {
        if (channel == null)
            return 8192;
        return channel.getPitchBend();
    }

    public int getPolyPressure(int noteNumber) {
        if (channel == null)
            return 0;
        return channel.getPolyPressure(noteNumber);
    }

    public int getProgram() {
        if (channel == null)
            return 0;
        return channel.getProgram();
    }

    public boolean getSolo() {
        if (channel == null)
            return false;
        return channel.getSolo();
    }

    public boolean localControl(boolean on) {
        if (channel == null)
            return false;
        return channel.localControl(on);
```

```
123     }
124
125     public void noteOff(int noteNumber) {
126         if (channel == null)
127             return;
128         channel.noteOff(noteNumber);
129     }
130
131     public void noteOff(int noteNumber, int velocity) {
132         if (channel == null)
133             return;
134         channel.noteOff(noteNumber, velocity);
135     }
136
137     public void noteOn(int noteNumber, int velocity) {
138         if (channel == null)
139             return;
140         channel.noteOn(noteNumber, velocity);
141     }
142
143     public void programChange(int program) {
144         if (channel == null)
145             return;
146         channel.programChange(program);
147     }
148
149     public void programChange(int bank, int program) {
150         if (channel == null)
151             return;
152         channel.programChange(bank, program);
153     }
154
155     public void resetAllControllers() {
156         if (channel == null)
157             return;
158         channel.resetAllControllers();
159     }
160
161     public void setChannelPressure(int pressure) {
162         if (channel == null)
163             return;
164         channel.setChannelPressure(pressure);
165     }
166
167     public void setMono(boolean on) {
168         if (channel == null)
169             return;
170         channel.setMono(on);
171     }
172
173     public void setMute(boolean mute) {
174         if (channel == null)
175             return;
176         channel.setMute(mute);
177     }
178
179     public void setOmni(boolean on) {
180         if (channel == null)
181             return;
182         channel.setOmni(on);
183     }
184
```

```
185     public void setPitchBend(int bend) {
186         if (channel == null)
187             return;
188         channel.setPitchBend(bend);
189     }
190 
191     public void setPolyPressure(int noteNumber, int pressure) {
192         if (channel == null)
193             return;
194         channel.setPolyPressure(noteNumber, pressure);
195     }
196 
197     public void setSolo(boolean soloState) {
198         if (channel == null)
199             return;
200         channel.setSolo(soloState);
201     }
202 }
```

83 com/sun/media/sound/SoftChorus.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.util.Arrays;

/**
 * A chorus effect made using LFO and variable delay. One for each channel
 * (left,right), with different starting phase for stereo effect.
 *
 * @author Karl Helgason
 */
public class SoftChorus implements SoftAudioProcessor {

    private static class VariableDelay {

        private float[] delaybuffer;
        private int rovepos = 0;
        private float gain = 1;
        private float rgain = 0;
        private float delay = 0;
        private float lastdelay = 0;
        private float feedback = 0;

        public VariableDelay(int maxbuffersize) {
            delaybuffer = new float[maxbuffersize];
        }

        public void setDelay(float delay) {
            this.delay = delay;
        }

        public void setFeedBack(float feedback) {
            this.feedback = feedback;
        }

        public void setGain(float gain) {
            this.gain = gain;
```

```
        }

        public void setReverbSendGain(float rgain) {
            this.rgain = rgain;
        }

        public void processMix(float[] in, float[] out, float[] rout) {
            float gain = this.gain;
            float delay = this.delay;
            float feedback = this.feedback;

            float[] delaybuffer = this.delaybuffer;
            int len = in.length;
            float delaydelta = (delay - lastdelay) / len;
            int rnlen = delaybuffer.length;
            int rovepos = this.rovepos;

            if (rout == null)
                for (int i = 0; i < len; i++) {
                    float r = rovepos - (lastdelay + 2) + rnlen;
                    int ri = (int) r;
                    float s = r - ri;
                    float a = delaybuffer[ri % rnlen];
                    float b = delaybuffer[(ri + 1) % rnlen];
                    float o = a * (1 - s) + b * (s);
                    out[i] += o * gain;
                    delaybuffer[rovepos] = in[i] + o * feedback;
                    rovepos = (rovepos + 1) % rnlen;
                    lastdelay += delaydelta;
                }
            else
                for (int i = 0; i < len; i++) {
                    float r = rovepos - (lastdelay + 2) + rnlen;
                    int ri = (int) r;
                    float s = r - ri;
                    float a = delaybuffer[ri % rnlen];
                    float b = delaybuffer[(ri + 1) % rnlen];
                    float o = a * (1 - s) + b * (s);
                    out[i] += o * gain;
                    rout[i] += o * rgain;
                    delaybuffer[rovepos] = in[i] + o * feedback;
                    rovepos = (rovepos + 1) % rnlen;
                    lastdelay += delaydelta;
                }
            this.rovepos = rovepos;
            lastdelay = delay;
        }

        public void processReplace(float[] in, float[] out, float[] rout) {
            Arrays.fill(out, 0);
            Arrays.fill(rout, 0);
            processMix(in, out, rout);
        }
    }

    private static class LFODelay {

        private double phase = 1;
        private double phase_step = 0;
        private double depth = 0;
        private VariableDelay vdelay;
        private double samplerate;
```

```
123         private double controlrate;
124 
125         public LFODelay(double samplerate, double controlrate) {
126             this.samplerate = samplerate;
127             this.controlrate = controlrate;
128             // vdelay = new VariableDelay((int)(samplerate*4));
129             vdelay = new VariableDelay((int) ((this.depth + 10) * 2));
130 
131         }
132 
133         public void setDepth(double depth) {
134             this.depth = depth * samplerate;
135             vdelay = new VariableDelay((int) ((this.depth + 10) * 2));
136         }
137 
138         public void setRate(double rate) {
139             double g = (Math.PI * 2) * (rate / controlrate);
140             phase_step = g;
141         }
142 
143         public void setPhase(double phase) {
144             this.phase = phase;
145         }
146 
147         public void setFeedBack(float feedback) {
148             vdelay.setFeedBack(feedback);
149         }
150 
151         public void setGain(float gain) {
152             vdelay.setGain(gain);
153         }
154 
155         public void setReverbSendGain(float rgain) {
156             vdelay.setReverbSendGain(rgain);
157         }
158 
159         public void processMix(float[] in, float[] out, float[] rout) {
160             phase += phase_step;
161             while(phase > (Math.PI * 2)) phase -= (Math.PI * 2);
162             vdelay.setDelay((float) (depth * 0.5 * (Math.cos(phase) + 2)));
163             vdelay.processMix(in, out, rout);
164         }
165 
166         public void processReplace(float[] in, float[] out, float[] rout) {
167             phase += phase_step;
168             while(phase > (Math.PI * 2)) phase -= (Math.PI * 2);
169             vdelay.setDelay((float) (depth * 0.5 * (Math.cos(phase) + 2)));
170             vdelay.processReplace(in, out, rout);
171 
172         }
173     }
174     private boolean mix = true;
175     private SoftAudioBuffer inputA;
176     private SoftAudioBuffer left;
177     private SoftAudioBuffer right;
178     private SoftAudioBuffer reverb;
179     private LFODelay vdelay1L;
180     private LFODelay vdelay1R;
181     private float rgain = 0;
182     private boolean dirty = true;
183     private double dirty_vdelay1L_rate;
184     private double dirty_vdelay1R_rate;
```

```
    private double dirty_vdelay1L_depth;
    private double dirty_vdelay1R_depth;
    private float dirty_vdelay1L_feedback;
    private float dirty_vdelay1R_feedback;
    private float dirty_vdelay1L_reverbsendgain;
    private float dirty_vdelay1R_reverbsendgain;
    private float controlrate;

    public void init(float samplerate, float controlrate) {
        this.controlrate = controlrate;
        vdelay1L = new LFODelay(samplerate, controlrate);
        vdelay1R = new LFODelay(samplerate, controlrate);
        vdelay1L.setGain(1.0f); // %
        vdelay1R.setGain(1.0f); // %
        vdelay1L.setPhase(0.5 * Math.PI);
        vdelay1R.setPhase(0);

        globalParameterControlChange(new int[]{0x01 * 128 + 0x02}, 0, 2);
    }

    public void globalParameterControlChange(int[] slothpath, long param,
            long value) {
        if (slothpath.length == 1) {
            if (slothpath[0] == 0x01 * 128 + 0x02) {
                if (param == 0) { // Chorus Type
                    switch ((int)value) {
                    case 0: // Chorus 1 0 (0%) 3 (0.4Hz) 5 (1.9ms) 0 (0%)
                        globalParameterControlChange(slothpath, 3, 0);
                        globalParameterControlChange(slothpath, 1, 3);
                        globalParameterControlChange(slothpath, 2, 5);
                        globalParameterControlChange(slothpath, 4, 0);
                        break;
                    case 1: // Chorus 2 5 (4%) 9 (1.1Hz) 19 (6.3ms) 0 (0%)
                        globalParameterControlChange(slothpath, 3, 5);
                        globalParameterControlChange(slothpath, 1, 9);
                        globalParameterControlChange(slothpath, 2, 19);
                        globalParameterControlChange(slothpath, 4, 0);
                        break;
                    case 2: // Chorus 3 8 (6%) 3 (0.4Hz) 19 (6.3ms) 0 (0%)
                        globalParameterControlChange(slothpath, 3, 8);
                        globalParameterControlChange(slothpath, 1, 3);
                        globalParameterControlChange(slothpath, 2, 19);
                        globalParameterControlChange(slothpath, 4, 0);
                        break;
                    case 3: // Chorus 4 16 (12%) 9 (1.1Hz) 16 (5.3ms) 0 (0%)
                        globalParameterControlChange(slothpath, 3, 16);
                        globalParameterControlChange(slothpath, 1, 9);
                        globalParameterControlChange(slothpath, 2, 16);
                        globalParameterControlChange(slothpath, 4, 0);
                        break;
                    case 4: // FB Chorus 64 (49%) 2 (0.2Hz) 24 (7.8ms) 0 (0%)
                        globalParameterControlChange(slothpath, 3, 64);
                        globalParameterControlChange(slothpath, 1, 2);
                        globalParameterControlChange(slothpath, 2, 24);
                        globalParameterControlChange(slothpath, 4, 0);
                        break;
                    case 5: // Flanger 112 (86%) 1 (0.1Hz) 5 (1.9ms) 0 (0%)
                        globalParameterControlChange(slothpath, 3, 112);
                        globalParameterControlChange(slothpath, 1, 1);
                        globalParameterControlChange(slothpath, 2, 5);
                        globalParameterControlChange(slothpath, 4, 0);
                        break;
```

```
                default:
                    break;
                }
            } else if (param == 1) { // Mod Rate
                dirty_vdelay1L_rate = (value * 0.122);
                dirty_vdelay1R_rate = (value * 0.122);
                dirty = true;
            } else if (param == 2) { // Mod Depth
                dirty_vdelay1L_depth = ((value + 1) / 3200.0);
                dirty_vdelay1R_depth = ((value + 1) / 3200.0);
                dirty = true;
            } else if (param == 3) { // Feedback
                dirty_vdelay1L_feedback = (value * 0.00763f);
                dirty_vdelay1R_feedback = (value * 0.00763f);
                dirty = true;
            }
            if (param == 4) { // Send to Reverb
                rgain = value * 0.00787f;
                dirty_vdelay1L_reverbsendgain = (value * 0.00787f);
                dirty_vdelay1R_reverbsendgain = (value * 0.00787f);
                dirty = true;
            }

        }
    }
}

public void processControlLogic() {
    if (dirty) {
        dirty = false;
        vdelay1L.setRate(dirty_vdelay1L_rate);
        vdelay1R.setRate(dirty_vdelay1R_rate);
        vdelay1L.setDepth(dirty_vdelay1L_depth);
        vdelay1R.setDepth(dirty_vdelay1R_depth);
        vdelay1L.setFeedBack(dirty_vdelay1L_feedback);
        vdelay1R.setFeedBack(dirty_vdelay1R_feedback);
        vdelay1L.setReverbSendGain(dirty_vdelay1L_reverbsendgain);
        vdelay1R.setReverbSendGain(dirty_vdelay1R_reverbsendgain);
    }
}
double silentcounter = 1000;

public void processAudio() {

    if (inputA.isSilent()) {
        silentcounter += 1 / controlrate;

        if (silentcounter > 1) {
            if (!mix) {
                left.clear();
                right.clear();
            }
            return;
        }
    } else
        silentcounter = 0;

    float[] inputA = this.inputA.array();
    float[] left = this.left.array();
    float[] right = this.right == null ? null : this.right.array();
    float[] reverb = rgain != 0 ? this.reverb.array() : null;

```

```
309         if (mix) {
310             vdelay1L.processMix(inputA, left, reverb);
311             if (right != null)
312                 vdelay1R.processMix(inputA, right, reverb);
313         } else {
314             vdelay1L.processReplace(inputA, left, reverb);
315             if (right != null)
316                 vdelay1R.processReplace(inputA, right, reverb);
317         }
318     }
319 
320     public void setInput(int pin, SoftAudioBuffer input) {
321         if (pin == 0)
322             inputA = input;
323     }
324 
325     public void setMixMode(boolean mix) {
326         this.mix = mix;
327     }
328 
329     public void setOutput(int pin, SoftAudioBuffer output) {
330         if (pin == 0)
331             left = output;
332         if (pin == 1)
333             right = output;
334         if (pin == 2)
335             reverb = output;
336     }
337 }
```

84 com/sun/media/sound/SoftControl.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * <code>SoftControl</code> are the basic controls
 * used for control-rate processing.
 *
 * @author Karl Helgason
 */
public interface SoftControl {

    public double[] get(int instance, String name);
}
```

85 com/sun/media/sound/SoftCubicResampler.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * A resampler that uses third-order (cubic) interpolation.
 *
 * @author Karl Helgason
 */
public class SoftCubicResampler extends SoftAbstractResampler {

    public int getPadding() {
        return 3;
    }

    public void interpolate(float[] in, float[] in_offset, float in_end,
            float[] startpitch, float pitchstep, float[] out, int[] out_offset,
            int out_end) {
        float pitch = startpitch[0];
        float ix = in_offset[0];
        int ox = out_offset[0];
        float ix_end = in_end;
        int ox_end = out_end;
        if (pitchstep == 0) {
            while (ix < ix_end && ox < ox_end) {
                int iix = (int) ix;
                float fix = ix - iix;
                float y0 = in[iix - 1];
                float y1 = in[iix];
                float y2 = in[iix + 1];
                float y3 = in[iix + 2];
                float a0 = y3 - y2 + y1 - y0;
                float a1 = y0 - y1 - a0;
                float a2 = y2 - y0;
                float a3 = y1;
                //float fix2 = fix * fix;
                //out[ox++] = (a0 * fix + a1) * fix2 + (a2 * fix + a3);
                out[ox++] = ((a0 * fix + a1) * fix + a2) * fix + a3;
```

```
                ix += pitch;
            }
        } else {
            while (ix < ix_end && ox < ox_end) {
                int iix = (int) ix;
                float fix = ix - iix;
                float y0 = in[iix - 1];
                float y1 = in[iix];
                float y2 = in[iix + 1];
                float y3 = in[iix + 2];
                float a0 = y3 - y2 + y1 - y0;
                float a1 = y0 - y1 - a0;
                float a2 = y2 - y0;
                float a3 = y1;
                //float fix2 = fix * fix;
                //out[ox++] = (a0 * fix + a1) * fix2 + (a2 * fix + a3);
                out[ox++] = ((a0 * fix + a1) * fix + a2) * fix + a3;
                ix += pitch;
                pitch += pitchstep;
            }
        }
        in_offset[0] = ix;
        out_offset[0] = ox;
        startpitch[0] = pitch;

    }
}
```

86 com/sun/media/sound/SoftEnvelopeGenerator.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * AHDSR control signal envelope generator.
 *
 * @author Karl Helgason
 */
public class SoftEnvelopeGenerator implements SoftProcess {

    public final static int EG_OFF = 0;
    public final static int EG_DELAY = 1;
    public final static int EG_ATTACK = 2;
    public final static int EG_HOLD = 3;
    public final static int EG_DECAY = 4;
    public final static int EG_SUSTAIN = 5;
    public final static int EG_RELEASE = 6;
    public final static int EG_SHUTDOWN = 7;
    public final static int EG_END = 8;
    int max_count = 10;
    int used_count = 0;
    private int[] stage = new int[max_count];
    private int[] stage_ix = new int[max_count];
    private double[] stage_v = new double[max_count];
    private int[] stage_count = new int[max_count];
    private double[][] on = new double[max_count][1];
    private double[][] active = new double[max_count][1];
    private double[][] out = new double[max_count][1];
    private double[][] delay = new double[max_count][1];
    private double[][] attack = new double[max_count][1];
    private double[][] hold = new double[max_count][1];
    private double[][] decay = new double[max_count][1];
    private double[][] sustain = new double[max_count][1];
    private double[][] release = new double[max_count][1];
    private double[][] shutdown = new double[max_count][1];
    private double[][] release2 = new double[max_count][1];
    private double[][] attack2 = new double[max_count][1];
```

```
    private double[][] decay2 = new double[max_count][1];
    private double control_time = 0;

    public void reset() {
        for (int i = 0; i < used_count; i++) {
            stage[i] = 0;
            on[i][0] = 0;
            out[i][0] = 0;
            delay[i][0] = 0;
            attack[i][0] = 0;
            hold[i][0] = 0;
            decay[i][0] = 0;
            sustain[i][0] = 0;
            release[i][0] = 0;
            shutdown[i][0] = 0;
            attack2[i][0] = 0;
            decay2[i][0] = 0;
            release2[i][0] = 0;
        }
        used_count = 0;
    }

    public void init(SoftSynthesizer synth) {
        control_time = 1.0 / synth.getControlRate();
        processControlLogic();
    }

    public double[] get(int instance, String name) {
        if (instance >= used_count)
            used_count = instance + 1;
        if (name == null)
            return out[instance];
        if (name.equals("on"))
            return on[instance];
        if (name.equals("active"))
            return active[instance];
        if (name.equals("delay"))
            return delay[instance];
        if (name.equals("attack"))
            return attack[instance];
        if (name.equals("hold"))
            return hold[instance];
        if (name.equals("decay"))
            return decay[instance];
        if (name.equals("sustain"))
            return sustain[instance];
        if (name.equals("release"))
            return release[instance];
        if (name.equals("shutdown"))
            return shutdown[instance];
        if (name.equals("attack2"))
            return attack2[instance];
        if (name.equals("decay2"))
            return decay2[instance];
        if (name.equals("release2"))
            return release2[instance];

        return null;
    }

    public void processControlLogic() {
        for (int i = 0; i < used_count; i++) {
```

```
123
124             if (stage[i] == EG_END)
125                 continue;
126
127             if ((stage[i] > EG_OFF) && (stage[i] < EG_RELEASE)) {
128                 if (on[i][0] < 0.5) {
129                     if (on[i][0] < -0.5) {
130                         stage_count[i] = (int)(Math.pow(2,
131                                 this.shutdown[i][0] / 1200.0) / control_time);
132                         if (stage_count[i] < 0)
133                             stage_count[i] = 0;
134                         stage_v[i] = out[i][0];
135                         stage_ix[i] = 0;
136                         stage[i] = EG_SHUTDOWN;
137                     } else {
138                         if ((release2[i][0] < 0.000001) && release[i][0] < 0
139                                 && Double.isInfinite(release[i][0])) {
140                             out[i][0] = 0;
141                             active[i][0] = 0;
142                             stage[i] = EG_END;
143                             continue;
144                         }
145
146                         stage_count[i] = (int)(Math.pow(2,
147                                 this.release[i][0] / 1200.0) / control_time);
148                         stage_count[i]
149                                 += (int)(this.release2[i][0]/(control_time * 1000));
150                         if (stage_count[i] < 0)
151                             stage_count[i] = 0;
152                         // stage_v[i] = out[i][0];
153                         stage_ix[i] = 0;
154
155                         double m = 1 - out[i][0];
156                         stage_ix[i] = (int)(stage_count[i] * m);
157
158                         stage[i] = EG_RELEASE;
159                     }
160                 }
161             }
162
163             switch (stage[i]) {
164             case EG_OFF:
165                 active[i][0] = 1;
166                 if (on[i][0] < 0.5)
167                     break;
168                 stage[i] = EG_DELAY;
169                 stage_ix[i] = (int)(Math.pow(2,
170                         this.delay[i][0] / 1200.0) / control_time);
171                 if (stage_ix[i] < 0)
172                     stage_ix[i] = 0;
173             case EG_DELAY:
174                 if (stage_ix[i] == 0) {
175                     double attack = this.attack[i][0];
176                     double attack2 = this.attack2[i][0];
177
178                     if (attack2 < 0.000001
179                             && (attack < 0 && Double.isInfinite(attack))) {
180                         out[i][0] = 1;
181                         stage[i] = EG_HOLD;
182                         stage_count[i] = (int)(Math.pow(2,
183                                 this.hold[i][0] / 1200.0) / control_time);
184                         stage_ix[i] = 0;
```

```
                } else {
                    stage[i] = EG_ATTACK;
                    stage_count[i] = (int)(Math.pow(2,
                            attack / 1200.0) / control_time);
                    stage_count[i] += (int)(attack2 / (control_time * 1000));
                    if (stage_count[i] < 0)
                        stage_count[i] = 0;
                    stage_ix[i] = 0;
                }
            } else
                stage_ix[i]--;
            break;
        case EG_ATTACK:
            stage_ix[i]++;
            if (stage_ix[i] >= stage_count[i]) {
                out[i][0] = 1;
                stage[i] = EG_HOLD;
            } else {
                // CONVEX attack
                double a = ((double)stage_ix[i]) / ((double)stage_count[i]);
                a = 1 + ((40.0 / 96.0) / Math.log(10)) * Math.log(a);
                if (a < 0)
                    a = 0;
                else if (a > 1)
                    a = 1;
                out[i][0] = a;
            }
            break;
        case EG_HOLD:
            stage_ix[i]++;
            if (stage_ix[i] >= stage_count[i]) {
                stage[i] = EG_DECAY;
                stage_count[i] = (int)(Math.pow(2,
                        this.decay[i][0] / 1200.0) / control_time);
                stage_count[i] += (int)(this.decay2[i][0]/(control_time*1000));
                if (stage_count[i] < 0)
                    stage_count[i] = 0;
                stage_ix[i] = 0;
            }
            break;
        case EG_DECAY:
            stage_ix[i]++;
            double sustain = this.sustain[i][0] * (1.0 / 1000.0);
            if (stage_ix[i] >= stage_count[i]) {
                out[i][0] = sustain;
                stage[i] = EG_SUSTAIN;
                if (sustain < 0.001) {
                    out[i][0] = 0;
                    active[i][0] = 0;
                    stage[i] = EG_END;
                }
            } else {
                double m = ((double)stage_ix[i]) / ((double)stage_count[i]);
                out[i][0] = (1 - m) + sustain * m;
            }
            break;
        case EG_SUSTAIN:
            break;
        case EG_RELEASE:
            stage_ix[i]++;
            if (stage_ix[i] >= stage_count[i]) {
                out[i][0] = 0;
```

```
                    active[i][0] = 0;
                    stage[i] = EG_END;
                } else {
                    double m = ((double)stage_ix[i]) / ((double)stage_count[i]);
                    out[i][0] = (1 - m); // *stage_v[i];

                    if (on[i][0] < -0.5) {
                        stage_count[i] = (int)(Math.pow(2,
                                this.shutdown[i][0] / 1200.0) / control_time);
                        if (stage_count[i] < 0)
                            stage_count[i] = 0;
                        stage_v[i] = out[i][0];
                        stage_ix[i] = 0;
                        stage[i] = EG_SHUTDOWN;
                    }

                    // re-damping
                    if (on[i][0] > 0.5) {
                        sustain = this.sustain[i][0] * (1.0 / 1000.0);
                        if (out[i][0] > sustain) {
                            stage[i] = EG_DECAY;
                            stage_count[i] = (int)(Math.pow(2,
                                    this.decay[i][0] / 1200.0) / control_time);
                            stage_count[i] +=
                                    (int)(this.decay2[i][0]/(control_time*1000));
                            if (stage_count[i] < 0)
                                stage_count[i] = 0;
                            m = (out[i][0] - 1) / (sustain - 1);
                            stage_ix[i] = (int) (stage_count[i] * m);
                        }
                    }

                }
                break;
            case EG_SHUTDOWN:
                stage_ix[i]++;
                if (stage_ix[i] >= stage_count[i]) {
                    out[i][0] = 0;
                    active[i][0] = 0;
                    stage[i] = EG_END;
                } else {
                    double m = ((double)stage_ix[i]) / ((double)stage_count[i]);
                    out[i][0] = (1 - m) * stage_v[i];
                }
                break;
            default:
                break;
            }
        }

    }
}
```

87 com/sun/media/sound/SoftFilter.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * Infinite impulse response (IIR) filter class.
 *
 * The filters where implemented and adapted using algorithms from musicdsp.org
 * archive: 1-RC and C filter, Simple 2-pole LP LP and HP filter, biquad,
 * tweaked butterworth RBJ Audio-EQ-Cookbook, EQ filter kookbook
 *
 * @author Karl Helgason
 */
public class SoftFilter {

    public final static int FILTERTYPE_LP6 = 0x00;
    public final static int FILTERTYPE_LP12 = 0x01;
    public final static int FILTERTYPE_HP12 = 0x11;
    public final static int FILTERTYPE_BP12 = 0x21;
    public final static int FILTERTYPE_NP12 = 0x31;
    public final static int FILTERTYPE_LP24 = 0x03;
    public final static int FILTERTYPE_HP24 = 0x13;

    //
    // 0x0 = 1st-order, 6 dB/oct
    // 0x1 = 2nd-order, 12 dB/oct
    // 0x2 = 3rd-order, 18 dB/oct
    // 0x3 = 4th-order, 24 db/oct
    //
    // 0x00 = LP, Low Pass Filter
    // 0x10 = HP, High Pass Filter
    // 0x20 = BP, Band Pass Filter
    // 0x30 = NP, Notch or Band Elimination Filter
    //
    private int filtertype = FILTERTYPE_LP6;
    private float samplerate;
    private float x1;
    private float x2;
```

```
    private float y1;
    private float y2;
    private float xx1;
    private float xx2;
    private float yy1;
    private float yy2;
    private float a0;
    private float a1;
    private float a2;
    private float b1;
    private float b2;
    private float q;
    private float gain = 1;
    private float wet = 0;
    private float last_wet = 0;
    private float last_a0;
    private float last_a1;
    private float last_a2;
    private float last_b1;
    private float last_b2;
    private float last_q;
    private float last_gain;
    private boolean last_set = false;
    private double cutoff = 44100;
    private double resonancedB = 0;
    private boolean dirty = true;

    public SoftFilter(float samplerate) {
        this.samplerate = samplerate;
        dirty = true;
    }

    public void setFrequency(double cent) {
        if (cutoff == cent)
            return;
        cutoff = cent;
        dirty = true;
    }

    public void setResonance(double db) {
        if (resonancedB == db)
            return;
        resonancedB = db;
        dirty = true;
    }

    public void reset() {
        dirty = true;
        last_set = false;
        x1 = 0;
        x2 = 0;
        y1 = 0;
        y2 = 0;
        xx1 = 0;
        xx2 = 0;
        yy1 = 0;
        yy2 = 0;
        wet = 0.0f;
        gain = 1.0f;
        a0 = 0;
        a1 = 0;
        a2 = 0;
```

```
123         b1 = 0;
124         b2 = 0;
125     }
126 
127     public void setFilterType(int filtertype) {
128         this.filtertype = filtertype;
129     }
130 
131     public void processAudio(SoftAudioBuffer sbuffer) {
132         if (filtertype == FILTERTYPE_LP6)
133             filter1(sbuffer);
134         if (filtertype == FILTERTYPE_LP12)
135             filter2(sbuffer);
136         if (filtertype == FILTERTYPE_HP12)
137             filter2(sbuffer);
138         if (filtertype == FILTERTYPE_BP12)
139             filter2(sbuffer);
140         if (filtertype == FILTERTYPE_NP12)
141             filter2(sbuffer);
142         if (filtertype == FILTERTYPE_LP24)
143             filter4(sbuffer);
144         if (filtertype == FILTERTYPE_HP24)
145             filter4(sbuffer);
146     }
147 
148     public void filter4(SoftAudioBuffer sbuffer) {
149 
150         float[] buffer = sbuffer.array();
151 
152         if (dirty) {
153             filter2calc();
154             dirty = false;
155         }
156         if (!last_set) {
157             last_a0 = a0;
158             last_a1 = a1;
159             last_a2 = a2;
160             last_b1 = b1;
161             last_b2 = b2;
162             last_gain = gain;
163             last_wet = wet;
164             last_set = true;
165         }
166 
167         if (wet > 0 || last_wet > 0) {
168 
169             int len = buffer.length;
170             float a0 = this.last_a0;
171             float a1 = this.last_a1;
172             float a2 = this.last_a2;
173             float b1 = this.last_b1;
174             float b2 = this.last_b2;
175             float gain = this.last_gain;
176             float wet = this.last_wet;
177             float a0_delta = (this.a0 - this.last_a0) / len;
178             float a1_delta = (this.a1 - this.last_a1) / len;
179             float a2_delta = (this.a2 - this.last_a2) / len;
180             float b1_delta = (this.b1 - this.last_b1) / len;
181             float b2_delta = (this.b2 - this.last_b2) / len;
182             float gain_delta = (this.gain - this.last_gain) / len;
183             float wet_delta = (this.wet - this.last_wet) / len;
184             float x1 = this.x1;
```

```
            float x2 = this.x2;
            float y1 = this.y1;
            float y2 = this.y2;
            float xx1 = this.xx1;
            float xx2 = this.xx2;
            float yy1 = this.yy1;
            float yy2 = this.yy2;

            if (wet_delta != 0) {
                for (int i = 0; i < len; i++) {
                    a0 += a0_delta;
                    a1 += a1_delta;
                    a2 += a2_delta;
                    b1 += b1_delta;
                    b2 += b2_delta;
                    gain += gain_delta;
                    wet += wet_delta;
                    float x = buffer[i];
                    float y = (a0*x + a1*x1 + a2*x2 - b1*y1 - b2*y2);
                    float xx = (y * gain) * wet + (x) * (1 - wet);
                    x2 = x1;
                    x1 = x;
                    y2 = y1;
                    y1 = y;
                    float yy = (a0*xx + a1*xx1 + a2*xx2 - b1*yy1 - b2*yy2);
                    buffer[i] = (yy * gain) * wet + (xx) * (1 - wet);
                    xx2 = xx1;
                    xx1 = xx;
                    yy2 = yy1;
                    yy1 = yy;
                }
            } else if (a0_delta == 0 && a1_delta == 0 && a2_delta == 0
                    && b1_delta == 0 && b2_delta == 0) {
                for (int i = 0; i < len; i++) {
                    float x = buffer[i];
                    float y = (a0*x + a1*x1 + a2*x2 - b1*y1 - b2*y2);
                    float xx = (y * gain) * wet + (x) * (1 - wet);
                    x2 = x1;
                    x1 = x;
                    y2 = y1;
                    y1 = y;
                    float yy = (a0*xx + a1*xx1 + a2*xx2 - b1*yy1 - b2*yy2);
                    buffer[i] = (yy * gain) * wet + (xx) * (1 - wet);
                    xx2 = xx1;
                    xx1 = xx;
                    yy2 = yy1;
                    yy1 = yy;
                }
            } else {
                for (int i = 0; i < len; i++) {
                    a0 += a0_delta;
                    a1 += a1_delta;
                    a2 += a2_delta;
                    b1 += b1_delta;
                    b2 += b2_delta;
                    gain += gain_delta;
                    float x = buffer[i];
                    float y = (a0*x + a1*x1 + a2*x2 - b1*y1 - b2*y2);
                    float xx = (y * gain) * wet + (x) * (1 - wet);
                    x2 = x1;
                    x1 = x;
                    y2 = y1;
```

```
                    y1 = y;
                    float yy = (a0*xx + a1*xx1 + a2*xx2 - b1*yy1 - b2*yy2);
                    buffer[i] = (yy * gain) * wet + (xx) * (1 - wet);
                    xx2 = xx1;
                    xx1 = xx;
                    yy2 = yy1;
                    yy1 = yy;
                }
            }

            if (Math.abs(x1) < 1.0E-8)
                x1 = 0;
            if (Math.abs(x2) < 1.0E-8)
                x2 = 0;
            if (Math.abs(y1) < 1.0E-8)
                y1 = 0;
            if (Math.abs(y2) < 1.0E-8)
                y2 = 0;
            this.x1 = x1;
            this.x2 = x2;
            this.y1 = y1;
            this.y2 = y2;
            this.xx1 = xx1;
            this.xx2 = xx2;
            this.yy1 = yy1;
            this.yy2 = yy2;
        }

        this.last_a0 = this.a0;
        this.last_a1 = this.a1;
        this.last_a2 = this.a2;
        this.last_b1 = this.b1;
        this.last_b2 = this.b2;
        this.last_gain = this.gain;
        this.last_wet = this.wet;

    }

    private double sinh(double x) {
        return (Math.exp(x) - Math.exp(-x)) * 0.5;
    }

    public void filter2calc() {

        double resonancedB = this.resonancedB;
        if (resonancedB < 0)
            resonancedB = 0;    // Negative dB are illegal.
        if (resonancedB > 30)
            resonancedB = 30;   // At least 22.5 dB is needed.
        if (filtertype == FILTERTYPE_LP24 || filtertype == FILTERTYPE_HP24)
            resonancedB *= 0.6;

        if (filtertype == FILTERTYPE_BP12) {
            wet = 1;
            double r = (cutoff / samplerate);
            if (r > 0.45)
                r = 0.45;

            double bandwidth = Math.PI * Math.pow(10.0, -(resonancedB / 20));

            double omega = 2 * Math.PI * r;
            double cs = Math.cos(omega);
```

```
            double sn = Math.sin(omega);
            double alpha = sn * sinh((Math.log(2)*bandwidth*omega) / (sn * 2));

            double b0 = alpha;
            double b1 = 0;
            double b2 = -alpha;
            double a0 = 1 + alpha;
            double a1 = -2 * cs;
            double a2 = 1 - alpha;

            double cf = 1.0 / a0;
            this.b1 = (float) (a1 * cf);
            this.b2 = (float) (a2 * cf);
            this.a0 = (float) (b0 * cf);
            this.a1 = (float) (b1 * cf);
            this.a2 = (float) (b2 * cf);
        }

        if (filtertype == FILTERTYPE_NP12) {
            wet = 1;
            double r = (cutoff / samplerate);
            if (r > 0.45)
                r = 0.45;

            double bandwidth = Math.PI * Math.pow(10.0, -(resonancedB / 20));

            double omega = 2 * Math.PI * r;
            double cs = Math.cos(omega);
            double sn = Math.sin(omega);
            double alpha = sn * sinh((Math.log(2)*bandwidth*omega) / (sn*2));

            double b0 = 1;
            double b1 = -2 * cs;
            double b2 = 1;
            double a0 = 1 + alpha;
            double a1 = -2 * cs;
            double a2 = 1 - alpha;

            double cf = 1.0 / a0;
            this.b1 = (float)(a1 * cf);
            this.b2 = (float)(a2 * cf);
            this.a0 = (float)(b0 * cf);
            this.a1 = (float)(b1 * cf);
            this.a2 = (float)(b2 * cf);
        }

        if (filtertype == FILTERTYPE_LP12 || filtertype == FILTERTYPE_LP24) {
            double r = (cutoff / samplerate);
            if (r > 0.45) {
                if (wet == 0) {
                    if (resonancedB < 0.00001)
                        wet = 0.0f;
                    else
                        wet = 1.0f;
                }
                r = 0.45;
            } else
                wet = 1.0f;

            double c = 1.0 / (Math.tan(Math.PI * r));
            double csq = c * c;
            double resonance = Math.pow(10.0, -(resonancedB / 20));
```

```
            double q = Math.sqrt(2.0f) * resonance;
            double a0 = 1.0 / (1.0 + (q * c) + (csq));
            double a1 = 2.0 * a0;
            double a2 = a0;
            double b1 = (2.0 * a0) * (1.0 - csq);
            double b2 = a0 * (1.0 - (q * c) + csq);

            this.a0 = (float)a0;
            this.a1 = (float)a1;
            this.a2 = (float)a2;
            this.b1 = (float)b1;
            this.b2 = (float)b2;

        }

        if (filtertype == FILTERTYPE_HP12 || filtertype == FILTERTYPE_HP24) {
            double r = (cutoff / samplerate);
            if (r > 0.45)
                r = 0.45;
            if (r < 0.0001)
                r = 0.0001;
            wet = 1.0f;
            double c = (Math.tan(Math.PI * (r)));
            double csq = c * c;
            double resonance = Math.pow(10.0, -(resonancedB / 20));
            double q = Math.sqrt(2.0f) * resonance;
            double a0 = 1.0 / (1.0 + (q * c) + (csq));
            double a1 = -2.0 * a0;
            double a2 = a0;
            double b1 = (2.0 * a0) * (csq - 1.0);
            double b2 = a0 * (1.0 - (q * c) + csq);

            this.a0 = (float)a0;
            this.a1 = (float)a1;
            this.a2 = (float)a2;
            this.b1 = (float)b1;
            this.b2 = (float)b2;

        }

    }

    public void filter2(SoftAudioBuffer sbuffer) {

        float[] buffer = sbuffer.array();

        if (dirty) {
            filter2calc();
            dirty = false;
        }
        if (!last_set) {
            last_a0 = a0;
            last_a1 = a1;
            last_a2 = a2;
            last_b1 = b1;
            last_b2 = b2;
            last_q = q;
            last_gain = gain;
            last_wet = wet;
            last_set = true;
        }

```

```
        if (wet > 0 || last_wet > 0) {

            int len = buffer.length;
            float a0 = this.last_a0;
            float a1 = this.last_a1;
            float a2 = this.last_a2;
            float b1 = this.last_b1;
            float b2 = this.last_b2;
            float gain = this.last_gain;
            float wet = this.last_wet;
            float a0_delta = (this.a0 - this.last_a0) / len;
            float a1_delta = (this.a1 - this.last_a1) / len;
            float a2_delta = (this.a2 - this.last_a2) / len;
            float b1_delta = (this.b1 - this.last_b1) / len;
            float b2_delta = (this.b2 - this.last_b2) / len;
            float gain_delta = (this.gain - this.last_gain) / len;
            float wet_delta = (this.wet - this.last_wet) / len;
            float x1 = this.x1;
            float x2 = this.x2;
            float y1 = this.y1;
            float y2 = this.y2;

            if (wet_delta != 0) {
                for (int i = 0; i < len; i++) {
                    a0 += a0_delta;
                    a1 += a1_delta;
                    a2 += a2_delta;
                    b1 += b1_delta;
                    b2 += b2_delta;
                    gain += gain_delta;
                    wet += wet_delta;
                    float x = buffer[i];
                    float y = (a0*x + a1*x1 + a2*x2 - b1*y1 - b2*y2);
                    buffer[i] = (y * gain) * wet + (x) * (1 - wet);
                    x2 = x1;
                    x1 = x;
                    y2 = y1;
                    y1 = y;
                }
            } else if (a0_delta == 0 && a1_delta == 0 && a2_delta == 0
                    && b1_delta == 0 && b2_delta == 0) {
                for (int i = 0; i < len; i++) {
                    float x = buffer[i];
                    float y = (a0*x + a1*x1 + a2*x2 - b1*y1 - b2*y2);
                    buffer[i] = y * gain;
                    x2 = x1;
                    x1 = x;
                    y2 = y1;
                    y1 = y;
                }
            } else {
                for (int i = 0; i < len; i++) {
                    a0 += a0_delta;
                    a1 += a1_delta;
                    a2 += a2_delta;
                    b1 += b1_delta;
                    b2 += b2_delta;
                    gain += gain_delta;
                    float x = buffer[i];
                    float y = (a0*x + a1*x1 + a2*x2 - b1*y1 - b2*y2);
                    buffer[i] = y * gain;
                    x2 = x1;
```

```
                    x1 = x;
                    y2 = y1;
                    y1 = y;
                }
            }

            if (Math.abs(x1) < 1.0E-8)
                x1 = 0;
            if (Math.abs(x2) < 1.0E-8)
                x2 = 0;
            if (Math.abs(y1) < 1.0E-8)
                y1 = 0;
            if (Math.abs(y2) < 1.0E-8)
                y2 = 0;
            this.x1 = x1;
            this.x2 = x2;
            this.y1 = y1;
            this.y2 = y2;
        }

        this.last_a0 = this.a0;
        this.last_a1 = this.a1;
        this.last_a2 = this.a2;
        this.last_b1 = this.b1;
        this.last_b2 = this.b2;
        this.last_q = this.q;
        this.last_gain = this.gain;
        this.last_wet = this.wet;

    }

    public void filter1calc() {
        if (cutoff < 120)
            cutoff = 120;
        double c = (7.0 / 6.0) * Math.PI * 2 * cutoff / samplerate;
        if (c > 1)
            c = 1;
        a0 = (float)(Math.sqrt(1 - Math.cos(c)) * Math.sqrt(0.5 * Math.PI));
        if (resonancedB < 0)
            resonancedB = 0;
        if (resonancedB > 20)
            resonancedB = 20;
        q = (float)(Math.sqrt(0.5) * Math.pow(10.0, -(resonancedB / 20)));
        gain = (float)Math.pow(10, -((resonancedB)) / 40.0);
        if (wet == 0.0f)
            if (resonancedB > 0.00001 || c < 0.9999999)
                wet = 1.0f;
    }

    public void filter1(SoftAudioBuffer sbuffer) {

        if (dirty) {
            filter1calc();
            dirty = false;
        }
        if (!last_set) {
            last_a0 = a0;
            last_q = q;
            last_gain = gain;
            last_wet = wet;
            last_set = true;
        }
```

```
557
558         if (wet > 0 || last_wet > 0) {
559
560             float[] buffer = sbuffer.array();
561             int len = buffer.length;
562             float a0 = this.last_a0;
563             float q = this.last_q;
564             float gain = this.last_gain;
565             float wet = this.last_wet;
566             float a0_delta = (this.a0 - this.last_a0) / len;
567             float q_delta = (this.q - this.last_q) / len;
568             float gain_delta = (this.gain - this.last_gain) / len;
569             float wet_delta = (this.wet - this.last_wet) / len;
570             float y2 = this.y2;
571             float y1 = this.y1;
572
573             if (wet_delta != 0) {
574                 for (int i = 0; i < len; i++) {
575                     a0 += a0_delta;
576                     q += q_delta;
577                     gain += gain_delta;
578                     wet += wet_delta;
579                     float ga0 = (1 - q * a0);
580                     y1 = ga0 * y1 + (a0) * (buffer[i] - y2);
581                     y2 = ga0 * y2 + (a0) * y1;
582                     buffer[i] = y2 * gain * wet + buffer[i] * (1 - wet);
583                 }
584             } else if (a0_delta == 0 && q_delta == 0) {
585                 float ga0 = (1 - q * a0);
586                 for (int i = 0; i < len; i++) {
587                     y1 = ga0 * y1 + (a0) * (buffer[i] - y2);
588                     y2 = ga0 * y2 + (a0) * y1;
589                     buffer[i] = y2 * gain;
590                 }
591             } else {
592                 for (int i = 0; i < len; i++) {
593                     a0 += a0_delta;
594                     q += q_delta;
595                     gain += gain_delta;
596                     float ga0 = (1 - q * a0);
597                     y1 = ga0 * y1 + (a0) * (buffer[i] - y2);
598                     y2 = ga0 * y2 + (a0) * y1;
599                     buffer[i] = y2 * gain;
600                 }
601             }
602
603             if (Math.abs(y2) < 1.0E-8)
604                 y2 = 0;
605             if (Math.abs(y1) < 1.0E-8)
606                 y1 = 0;
607             this.y2 = y2;
608             this.y1 = y1;
609         }
610
611         this.last_a0 = this.a0;
612         this.last_q = this.q;
613         this.last_gain = this.gain;
614         this.last_wet = this.wet;
615     }
616 }
```

88 com/sun/media/sound/SoftInstrument.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import javax.sound.midi.Instrument;
import javax.sound.midi.MidiChannel;

/**
 * Software synthesizer internal instrument.
 *
 * @author Karl Helgason
 */
public class SoftInstrument extends Instrument {

    private SoftPerformer[] performers;
    private ModelPerformer[] modelperformers;
    private Object data;
    private ModelInstrument ins;

    public SoftInstrument(ModelInstrument ins) {
        super(ins.getSoundbank(), ins.getPatch(), ins.getName(),
                ins.getDataClass());
        data = ins.getData();
        this.ins = ins;
        initPerformers(((ModelInstrument)ins).getPerformers());
    }

    public SoftInstrument(ModelInstrument ins,
            ModelPerformer[] overrideperformers) {
        super(ins.getSoundbank(), ins.getPatch(), ins.getName(),
                ins.getDataClass());
        data = ins.getData();
        this.ins = ins;
        initPerformers(overrideperformers);
    }

    private void initPerformers(ModelPerformer[] modelperformers) {
        this.modelperformers = modelperformers;
```

```
61          performers = new SoftPerformer[modelperformers.length];
62          for (int i = 0; i < modelperformers.length; i++)
63              performers[i] = new SoftPerformer(modelperformers[i]);
64      }
65
66      public ModelDirector getDirector(MidiChannel channel,
67              ModelDirectedPlayer player) {
68          return ins.getDirector(modelperformers, channel, player);
69      }
70
71      public ModelInstrument getSourceInstrument() {
72          return ins;
73      }
74
75      public Object getData() {
76          return data;
77      }
78
79      public SoftPerformer[] getPerformers() {
80          return performers;
81      }
82  }
```

89 com/sun/media/sound/SoftJitterCorrector.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.io.EOFException;
import java.io.IOException;
import java.io.InputStream;

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

/**
 * A jitter corrector to be used with SoftAudioPusher.
 *
 * @author Karl Helgason
 */
public class SoftJitterCorrector extends AudioInputStream {

    private static class JitterStream extends InputStream {

        static int MAX_BUFFER_SIZE = 1048576;
        boolean active = true;
        Thread thread;
        AudioInputStream stream;
        // Cyclic buffer
        int writepos = 0;
        int readpos = 0;
        byte[][] buffers;
        Object buffers_mutex = new Object();

        // Adapative Drift Statistics
        int w_count = 1000;
        int w_min_tol = 2;
        int w_max_tol = 10;
        int w = 0;
        int w_min = -1;
        // Current read buffer
        int bbuffer_pos = 0;
```

```
        int bbuffer_max = 0;
        byte[] bbuffer = null;

        public byte[] nextReadBuffer() {
            synchronized (buffers_mutex) {
                if (writepos > readpos) {
                    int w_m = writepos - readpos;
                    if (w_m < w_min)
                        w_min = w_m;

                    int buffpos = readpos;
                    readpos++;
                    return buffers[buffpos % buffers.length];
                }
                w_min = -1;
                w = w_count - 1;
            }
            while (true) {
                try {
                    Thread.sleep(1);
                } catch (InterruptedException e) {
                    //e.printStackTrace();
                    return null;
                }
                synchronized (buffers_mutex) {
                    if (writepos > readpos) {
                        w = 0;
                        w_min = -1;
                        w = w_count - 1;
                        int buffpos = readpos;
                        readpos++;
                        return buffers[buffpos % buffers.length];
                    }
                }
            }
        }

        public byte[] nextWriteBuffer() {
            synchronized (buffers_mutex) {
                return buffers[writepos % buffers.length];
            }
        }

        public void commit() {
            synchronized (buffers_mutex) {
                writepos++;
                if ((writepos - readpos) > buffers.length) {
                    int newsize = (writepos - readpos) + 10;
                    newsize = Math.max(buffers.length * 2, newsize);
                    buffers = new byte[newsize][buffers[0].length];
                }
            }
        }

        public JitterStream(AudioInputStream s, int buffersize,
                int smallbuffersize) {
            this.w_count = 10 * (buffersize / smallbuffersize);
            if (w_count < 100)
                w_count = 100;
            this.buffers
                    = new byte[(buffersize/smallbuffersize)+10][smallbuffersize];
            this.bbuffer_max = MAX_BUFFER_SIZE / smallbuffersize;
```

```
123             this.stream = s;
124
125
126             Runnable runnable = new Runnable() {
127
128                 public void run() {
129                     AudioFormat format = stream.getFormat();
130                     int bufflen = buffers[0].length;
131                     int frames = bufflen / format.getFrameSize();
132                     long nanos = (long) (frames * 1000000000.0
133                                          / format.getSampleRate());
134                     long now = System.nanoTime();
135                     long next = now + nanos;
136                     int correction = 0;
137                     while (true) {
138                         synchronized (JitterStream.this) {
139                             if (!active)
140                                 break;
141                         }
142                         int curbuffsize;
143                         synchronized (buffers) {
144                             curbuffsize = writepos - readpos;
145                             if (correction == 0) {
146                                 w++;
147                                 if (w_min != Integer.MAX_VALUE) {
148                                     if (w == w_count) {
149                                         correction = 0;
150                                         if (w_min < w_min_tol) {
151                                             correction = (w_min_tol + w_max_tol)
152                                                          / 2 - w_min;
153                                         }
154                                         if (w_min > w_max_tol) {
155                                             correction = (w_min_tol + w_max_tol)
156                                                          / 2 - w_min;
157                                         }
158                                         w = 0;
159                                         w_min = Integer.MAX_VALUE;
160                                     }
161                                 }
162                             }
163                         }
164                         while (curbuffsize > bbuffer_max) {
165                             synchronized (buffers) {
166                                 curbuffsize = writepos - readpos;
167                             }
168                             synchronized (JitterStream.this) {
169                                 if (!active)
170                                     break;
171                             }
172                             try {
173                                 Thread.sleep(1);
174                             } catch (InterruptedException e) {
175                                 //e.printStackTrace();
176                             }
177                         }
178
179                         if (correction < 0)
180                             correction++;
181                         else {
182                             byte[] buff = nextWriteBuffer();
183                             try {
184                                 int n = 0;
```

```
                            while (n != buff.length) {
                                int s = stream.read(buff, n, buff.length
                                        - n);
                                if (s < 0)
                                    throw new EOFException();
                                if (s == 0)
                                    Thread.yield();
                                n += s;
                            }
                        } catch (IOException e1) {
                            //e1.printStackTrace();
                        }
                        commit();
                    }

                    if (correction > 0) {
                        correction--;
                        next = System.nanoTime() + nanos;
                        continue;
                    }
                    long wait = next - System.nanoTime();
                    if (wait > 0) {
                        try {
                            Thread.sleep(wait / 1000000L);
                        } catch (InterruptedException e) {
                            //e.printStackTrace();
                        }
                    }
                    next += nanos;
                }
            }
        };

        thread = new Thread(runnable);
        thread.setDaemon(true);
        thread.setPriority(Thread.MAX_PRIORITY);
        thread.start();
    }

    public void close() throws IOException {
        synchronized (this) {
            active = false;
        }
        try {
            thread.join();
        } catch (InterruptedException e) {
            //e.printStackTrace();
        }
        stream.close();
    }

    public int read() throws IOException {
        byte[] b = new byte[1];
        if (read(b) == -1)
            return -1;
        return b[0] & 0xFF;
    }

    public void fillBuffer() {
        bbuffer = nextReadBuffer();
        bbuffer_pos = 0;
    }
```

```
247
248         public int read(byte[] b, int off, int len) {
249             if (bbuffer == null)
250                 fillBuffer();
251             int bbuffer_len = bbuffer.length;
252             int offlen = off + len;
253             while (off < offlen) {
254                 if (available() == 0)
255                     fillBuffer();
256                 else {
257                     byte[] bbuffer = this.bbuffer;
258                     int bbuffer_pos = this.bbuffer_pos;
259                     while (off < offlen && bbuffer_pos < bbuffer_len)
260                         b[off++] = bbuffer[bbuffer_pos++];
261                     this.bbuffer_pos = bbuffer_pos;
262                 }
263             }
264             return len;
265         }
266
267         public int available() {
268             return bbuffer.length - bbuffer_pos;
269         }
270     }
271
272     public SoftJitterCorrector(AudioInputStream stream, int buffersize,
273             int smallbuffersize) {
274         super(new JitterStream(stream, buffersize, smallbuffersize),
275                 stream.getFormat(), stream.getFrameLength());
276     }
277 }
```

90 com/sun/media/sound/SoftLanczosResampler.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * Lanczos interpolation resampler.
 *
 * @author Karl Helgason
 */
public class SoftLanczosResampler extends SoftAbstractResampler {

    float[][] sinc_table;
    int sinc_table_fsize = 2000;
    int sinc_table_size = 5;
    int sinc_table_center = sinc_table_size / 2;

    public SoftLanczosResampler() {
        super();
        sinc_table = new float[sinc_table_fsize][];
        for (int i = 0; i < sinc_table_fsize; i++) {
            sinc_table[i] = sincTable(sinc_table_size, -i
                            / ((float) sinc_table_fsize));
        }
    }

    // Normalized sinc function
    public static double sinc(double x) {
        return (x == 0.0) ? 1.0 : Math.sin(Math.PI * x) / (Math.PI * x);
    }

    // Generate sinc table
    public static float[] sincTable(int size, float offset) {
        int center = size / 2;
        float[] w = new float[size];
        for (int k = 0; k < size; k++) {
            float x = (-center + k + offset);
            if (x < -2 || x > 2)
                w[k] = 0;
```

```
            else if (x == 0)
                w[k] = 1;
            else {
                w[k] = (float)(2.0 * Math.sin(Math.PI * x)
                               * Math.sin(Math.PI * x / 2.0)
                               / ((Math.PI * x) * (Math.PI * x)));
            }
        }
        return w;
    }

    public int getPadding() // must be at least half of sinc_table_size
    {
        return sinc_table_size / 2 + 2;
    }

    public void interpolate(float[] in, float[] in_offset, float in_end,
            float[] startpitch, float pitchstep, float[] out, int[] out_offset,
            int out_end) {
        float pitch = startpitch[0];
        float ix = in_offset[0];
        int ox = out_offset[0];
        float ix_end = in_end;
        int ox_end = out_end;

        if (pitchstep == 0) {
            while (ix < ix_end && ox < ox_end) {
                int iix = (int) ix;
                float[] sinc_table
                        = this.sinc_table[(int) ((ix - iix) * sinc_table_fsize)];
                int xx = iix - sinc_table_center;
                float y = 0;
                for (int i = 0; i < sinc_table_size; i++, xx++)
                    y += in[xx] * sinc_table[i];
                out[ox++] = y;
                ix += pitch;
            }
        } else {
            while (ix < ix_end && ox < ox_end) {
                int iix = (int) ix;
                float[] sinc_table
                        = this.sinc_table[(int) ((ix - iix) * sinc_table_fsize)];
                int xx = iix - sinc_table_center;
                float y = 0;
                for (int i = 0; i < sinc_table_size; i++, xx++)
                    y += in[xx] * sinc_table[i];
                out[ox++] = y;

                ix += pitch;
                pitch += pitchstep;
            }
        }
        in_offset[0] = ix;
        out_offset[0] = ox;
        startpitch[0] = pitch;

    }
}
```

91 com/sun/media/sound/SoftLimiter.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * A simple look-ahead volume limiter with very fast attack and fast release.
 * This filter is used for preventing clipping.
 *
 * @author Karl Helgason
 */
public class SoftLimiter implements SoftAudioProcessor {

    float lastmax = 0;
    float gain = 1;
    float[] temp_bufferL;
    float[] temp_bufferR;
    boolean mix = false;
    SoftAudioBuffer bufferL;
    SoftAudioBuffer bufferR;
    SoftAudioBuffer bufferLout;
    SoftAudioBuffer bufferRout;
    float controlrate;

    public void init(float samplerate, float controlrate) {
        this.controlrate = controlrate;
    }

    public void setInput(int pin, SoftAudioBuffer input) {
        if (pin == 0)
            bufferL = input;
        if (pin == 1)
            bufferR = input;
    }

    public void setOutput(int pin, SoftAudioBuffer output) {
        if (pin == 0)
            bufferLout = output;
        if (pin == 1)
```

```
61              bufferRout = output;
62      }
63
64      public void setMixMode(boolean mix) {
65          this.mix = mix;
66      }
67
68      public void globalParameterControlChange(int[] slothpath, long param,
69              long value) {
70      }
71
72      double silentcounter = 0;
73
74      public void processAudio() {
75          if (this.bufferL.isSilent()
76                  && (this.bufferR == null || this.bufferR.isSilent())) {
77              silentcounter += 1 / controlrate;
78
79              if (silentcounter > 60) {
80                  if (!mix) {
81                      bufferLout.clear();
82                      if(bufferRout != null) bufferRout.clear();
83                  }
84                  return;
85              }
86          } else
87              silentcounter = 0;
88
89          float[] bufferL = this.bufferL.array();
90          float[] bufferR = this.bufferR == null ? null : this.bufferR.array();
91          float[] bufferLout = this.bufferLout.array();
92          float[] bufferRout = this.bufferRout == null
93                                  ? null : this.bufferRout.array();
94
95          if (temp_bufferL == null || temp_bufferL.length < bufferL.length)
96              temp_bufferL = new float[bufferL.length];
97          if (bufferR != null)
98              if (temp_bufferR == null || temp_bufferR.length < bufferR.length)
99                  temp_bufferR = new float[bufferR.length];
100
101         float max = 0;
102         int len = bufferL.length;
103
104         if (bufferR == null) {
105             for (int i = 0; i < len; i++) {
106                 if (bufferL[i] > max)
107                     max = bufferL[i];
108                 if (-bufferL[i] > max)
109                     max = -bufferL[i];
110             }
111         } else {
112             for (int i = 0; i < len; i++) {
113                 if (bufferL[i] > max)
114                     max = bufferL[i];
115                 if (bufferR[i] > max)
116                     max = bufferR[i];
117                 if (-bufferL[i] > max)
118                     max = -bufferL[i];
119                 if (-bufferR[i] > max)
120                     max = -bufferR[i];
121             }
122         }
```

```
123
124         float lmax = lastmax;
125         lastmax = max;
126         if (lmax > max)
127             max = lmax;
128
129         float newgain = 1;
130         if (max > 0.99f)
131             newgain = 0.99f / max;
132         else
133             newgain = 1;
134
135         if (newgain > gain)
136             newgain = (newgain + gain * 9) / 10f;
137
138         float gaindelta = (newgain - gain) / len;
139         if (mix) {
140             if (bufferR == null) {
141                 for (int i = 0; i < len; i++) {
142                     gain += gaindelta;
143                     float bL = bufferL[i];
144                     float tL = temp_bufferL[i];
145                     temp_bufferL[i] = bL;
146                     bufferLout[i] += tL * gain;
147                 }
148             } else {
149                 for (int i = 0; i < len; i++) {
150                     gain += gaindelta;
151                     float bL = bufferL[i];
152                     float bR = bufferR[i];
153                     float tL = temp_bufferL[i];
154                     float tR = temp_bufferR[i];
155                     temp_bufferL[i] = bL;
156                     temp_bufferR[i] = bR;
157                     bufferLout[i] += tL * gain;
158                     bufferRout[i] += tR * gain;
159                 }
160             }
161
162         } else {
163             if (bufferR == null) {
164                 for (int i = 0; i < len; i++) {
165                     gain += gaindelta;
166                     float bL = bufferL[i];
167                     float tL = temp_bufferL[i];
168                     temp_bufferL[i] = bL;
169                     bufferLout[i] = tL * gain;
170                 }
171             } else {
172                 for (int i = 0; i < len; i++) {
173                     gain += gaindelta;
174                     float bL = bufferL[i];
175                     float bR = bufferR[i];
176                     float tL = temp_bufferL[i];
177                     float tR = temp_bufferR[i];
178                     temp_bufferL[i] = bL;
179                     temp_bufferR[i] = bR;
180                     bufferLout[i] = tL * gain;
181                     bufferRout[i] = tR * gain;
182                 }
183             }
184
```

```
185             }
186             gain = newgain;
187         }
188 
189         public void processControlLogic() {
190         }
191 }
```

92 com/sun/media/sound/SoftLinearResampler.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * A resampler that uses first-order (linear) interpolation.
 *
 * @author Karl Helgason
 */
public class SoftLinearResampler extends SoftAbstractResampler {

    public int getPadding() {
        return 2;
    }

    public void interpolate(float[] in, float[] in_offset, float in_end,
            float[] startpitch, float pitchstep, float[] out, int[] out_offset,
            int out_end) {

        float pitch = startpitch[0];
        float ix = in_offset[0];
        int ox = out_offset[0];
        float ix_end = in_end;
        int ox_end = out_end;
        if (pitchstep == 0f) {
            while (ix < ix_end && ox < ox_end) {
                int iix = (int) ix;
                float fix = ix - iix;
                float i = in[iix];
                out[ox++] = i + (in[iix + 1] - i) * fix;
                ix += pitch;
            }
        } else {
            while (ix < ix_end && ox < ox_end) {
                int iix = (int) ix;
                float fix = ix - iix;
                float i = in[iix];
                out[ox++] = i + (in[iix + 1] - i) * fix;
```

```
                ix += pitch;
                pitch += pitchstep;
            }
        }
        in_offset[0] = ix;
        out_offset[0] = ox;
        startpitch[0] = pitch;

    }
}
```

93 com/sun/media/sound/SoftLinearResampler2.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * A resampler that uses first-order (linear) interpolation.
 *
 * This one doesn't perform float to int casting inside the processing loop.
 *
 * @author Karl Helgason
 */
public class SoftLinearResampler2 extends SoftAbstractResampler {

    public int getPadding() {
        return 2;
    }

    public void interpolate(float[] in, float[] in_offset, float in_end,
            float[] startpitch, float pitchstep, float[] out, int[] out_offset,
            int out_end) {

        float pitch = startpitch[0];
        float ix = in_offset[0];
        int ox = out_offset[0];
        float ix_end = in_end;
        int ox_end = out_end;

        // Check if we have do anything
        if (!(ix < ix_end && ox < ox_end))
            return;

        // 15 bit shift was choosed because
        // it resulted in no drift between p_ix and ix.
        int p_ix = (int) (ix * (1 << 15));
        int p_ix_end = (int) (ix_end * (1 << 15));
        int p_pitch = (int) (pitch * (1 << 15));
        // Pitch needs to recalculated
        // to ensure no drift between p_ix and ix.
```

```
        pitch = p_pitch * (1f / (1 << 15));

        if (pitchstep == 0f) {

            // To reduce
            //    while (p_ix < p_ix_end && ox < ox_end)
            // into
            //    while  (ox < ox_end)
            // We need to calculate new ox_end value.
            int p_ix_len = p_ix_end - p_ix;
            int p_mod = p_ix_len % p_pitch;
            if (p_mod != 0)
                p_ix_len += p_pitch - p_mod;
            int ox_end2 = ox + p_ix_len / p_pitch;
            if (ox_end2 < ox_end)
                ox_end = ox_end2;

            while (ox < ox_end) {
                int iix = p_ix >> 15;
                float fix = ix - iix;
                float i = in[iix];
                out[ox++] = i + (in[iix + 1] - i) * fix;
                p_ix += p_pitch;
                ix += pitch;
            }

        } else {

            int p_pitchstep = (int) (pitchstep * (1 << 15));
            pitchstep = p_pitchstep * (1f / (1 << 15));

            while (p_ix < p_ix_end && ox < ox_end) {
                int iix = p_ix >> 15;
                float fix = ix - iix;
                float i = in[iix];
                out[ox++] = i + (in[iix + 1] - i) * fix;
                ix += pitch;
                p_ix += p_pitch;
                pitch += pitchstep;
                p_pitch += p_pitchstep;
            }
        }
        in_offset[0] = ix;
        out_offset[0] = ox;
        startpitch[0] = pitch;

    }
}
```

94 com/sun/media/sound/SoftLowFrequencyOscillator.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * LFO control signal generator.
 *
 * @author Karl Helgason
 */
public class SoftLowFrequencyOscillator implements SoftProcess {

    private int max_count = 10;
    private int used_count = 0;
    private double[][] out = new double[max_count][1];
    private double[][] delay = new double[max_count][1];
    private double[][] delay2 = new double[max_count][1];
    private double[][] freq = new double[max_count][1];
    private int[] delay_counter = new int[max_count];
    private double[] sin_phase = new double[max_count];
    private double[] sin_stepfreq = new double[max_count];
    private double[] sin_step = new double[max_count];
    private double control_time = 0;
    private double sin_factor = 0;
    private static double PI2 = 2.0 * Math.PI;

    public SoftLowFrequencyOscillator() {
        // If sin_step is 0 then sin_stepfreq must be -INF
        for (int i = 0; i < sin_stepfreq.length; i++) {
            sin_stepfreq[i] = Double.NEGATIVE_INFINITY;
        }
    }

    public void reset() {
        for (int i = 0; i < used_count; i++) {
            out[i][0] = 0;
            delay[i][0] = 0;
            delay2[i][0] = 0;
            freq[i][0] = 0;
```

```
            delay_counter[i] = 0;
            sin_phase[i] = 0;
            // If sin_step is 0 then sin_stepfreq must be -INF
            sin_stepfreq[i] = Double.NEGATIVE_INFINITY;
            sin_step[i] = 0;
        }
        used_count = 0;
    }

    public void init(SoftSynthesizer synth) {
        control_time = 1.0 / synth.getControlRate();
        sin_factor = control_time * 2 * Math.PI;
        for (int i = 0; i < used_count; i++) {
            delay_counter[i] = (int)(Math.pow(2,
                    this.delay[i][0] / 1200.0) / control_time);
            delay_counter[i] += (int)(delay2[i][0] / (control_time * 1000));
        }
        processControlLogic();
    }

    public void processControlLogic() {
        for (int i = 0; i < used_count; i++) {
            if (delay_counter[i] > 0) {
                delay_counter[i]--;
                out[i][0] = 0.5;
            } else {
                double f = freq[i][0];

                if (sin_stepfreq[i] != f) {
                    sin_stepfreq[i] = f;
                    double fr = 440.0 * Math.exp(
                            (f - 6900.0) * (Math.log(2) / 1200.0));
                    sin_step[i] = fr * sin_factor;
                }
                /*
                double fr = 440.0 * Math.pow(2.0,
                (freq[i][0] - 6900.0) / 1200.0);
                sin_phase[i] += fr * sin_factor;
                 */
                /*
                sin_phase[i] += sin_step[i];
                while (sin_phase[i] > PI2)
                sin_phase[i] -= PI2;
                out[i][0] = 0.5 + Math.sin(sin_phase[i]) * 0.5;
                 */
                double p = sin_phase[i];
                p += sin_step[i];
                while (p > PI2)
                    p -= PI2;
                out[i][0] = 0.5 + Math.sin(p) * 0.5;
                sin_phase[i] = p;

            }
        }
    }

    public double[] get(int instance, String name) {
        if (instance >= used_count)
            used_count = instance + 1;
        if (name == null)
            return out[instance];
        if (name.equals("delay"))
```

```
            return delay[instance];
        if (name.equals("delay2"))
            return delay2[instance];
        if (name.equals("freq"))
            return freq[instance];
        return null;
    }
}
```

95 com/sun/media/sound/SoftMainMixer.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.io.IOException;
import java.io.InputStream;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;
import java.util.TreeMap;
import java.util.Map.Entry;

import javax.sound.midi.MidiMessage;
import javax.sound.midi.Patch;
import javax.sound.midi.ShortMessage;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;

/**
 * Software synthesizer main audio mixer.
 *
 * @author Karl Helgason
 */
public class SoftMainMixer {

    // A private class thats contains a ModelChannelMixer and it's private buffers.
    // This becomes necessary when we want to have separate delay buffers for each channel mixer.
    private class SoftChannelMixerContainer
    {
        ModelChannelMixer mixer;
        SoftAudioBuffer[] buffers;
    }

    public final static int CHANNEL_LEFT = 0;
    public final static int CHANNEL_RIGHT = 1;
    public final static int CHANNEL_MONO = 2;
    public final static int CHANNEL_DELAY_LEFT = 3;
    public final static int CHANNEL_DELAY_RIGHT = 4;
```

```
    public final static int CHANNEL_DELAY_MONO = 5;
    public final static int CHANNEL_EFFECT1 = 6;
    public final static int CHANNEL_EFFECT2 = 7;
    public final static int CHANNEL_DELAY_EFFECT1 = 8;
    public final static int CHANNEL_DELAY_EFFECT2 = 9;
    public final static int CHANNEL_LEFT_DRY = 10;
    public final static int CHANNEL_RIGHT_DRY = 11;
    public final static int CHANNEL_SCRATCH1 = 12;
    public final static int CHANNEL_SCRATCH2 = 13;
    protected boolean active_sensing_on = false;
    private long msec_last_activity = -1;
    private boolean pusher_silent = false;
    private int pusher_silent_count = 0;
    private long sample_pos = 0;
    protected boolean readfully = true;
    private Object control_mutex;
    private SoftSynthesizer synth;
    private float samplerate = 44100;
    private int nrofchannels = 2;
    private SoftVoice[] voicestatus = null;
    private SoftAudioBuffer[] buffers;
    private SoftReverb reverb;
    private SoftAudioProcessor chorus;
    private SoftAudioProcessor agc;
    private long msec_buffer_len = 0;
    private int buffer_len = 0;
    protected TreeMap<Long, Object> midimessages = new TreeMap<Long, Object>();
    private int delay_midievent = 0;
    private int max_delay_midievent = 0;
    double last_volume_left = 1.0;
    double last_volume_right = 1.0;
    private double[] co_master_balance = new double[1];
    private double[] co_master_volume = new double[1];
    private double[] co_master_coarse_tuning = new double[1];
    private double[] co_master_fine_tuning = new double[1];
    private AudioInputStream ais;
    private Set<SoftChannelMixerContainer> registeredMixers = null;
    private Set<ModelChannelMixer> stoppedMixers = null;
    private SoftChannelMixerContainer[] cur_registeredMixers = null;
    protected SoftControl co_master = new SoftControl() {

        double[] balance = co_master_balance;
        double[] volume = co_master_volume;
        double[] coarse_tuning = co_master_coarse_tuning;
        double[] fine_tuning = co_master_fine_tuning;

        public double[] get(int instance, String name) {
            if (name == null)
                return null;
            if (name.equals("balance"))
                return balance;
            if (name.equals("volume"))
                return volume;
            if (name.equals("coarse_tuning"))
                return coarse_tuning;
            if (name.equals("fine_tuning"))
                return fine_tuning;
            return null;
        }
    };

    private void processSystemExclusiveMessage(byte[] data) {
```

```
        synchronized (synth.control_mutex) {
            activity();

            // Universal Non-Real-Time SysEx
            if ((data[1] & 0xFF) == 0x7E) {
                int deviceID = data[2] & 0xFF;
                if (deviceID == 0x7F || deviceID == synth.getDeviceID()) {
                    int subid1 = data[3] & 0xFF;
                    int subid2;
                    switch (subid1) {
                    case 0x08:  // MIDI Tuning Standard
                        subid2 = data[4] & 0xFF;
                        switch (subid2) {
                        case 0x01:  // BULK TUNING DUMP
                        {
                            // http://www.midi.org/about-midi/tuning.shtml
                            SoftTuning tuning = synth.getTuning(new Patch(0,
                                    data[5] & 0xFF));
                            tuning.load(data);
                            break;
                        }
                        case 0x04:  // KEY-BASED TUNING DUMP
                        case 0x05:  // SCALE/OCTAVE TUNING DUMP, 1 byte format
                        case 0x06:  // SCALE/OCTAVE TUNING DUMP, 2 byte format
                        case 0x07:  // SINGLE NOTE TUNING CHANGE (NON REAL-TIME)
                                    // (BANK)
                        {
                            // http://www.midi.org/about-midi/tuning_extens.shtml
                            SoftTuning tuning = synth.getTuning(new Patch(
                                    data[5] & 0xFF, data[6] & 0xFF));
                            tuning.load(data);
                            break;
                        }
                        case 0x08:  // scale/octave tuning 1-byte form (Non
                                    // Real-Time)
                        case 0x09:  // scale/octave tuning 2-byte form (Non
                                    // Real-Time)
                        {
                            // http://www.midi.org/about-midi/tuning-scale.shtml
                            SoftTuning tuning = new SoftTuning(data);
                            int channelmask = (data[5] & 0xFF) * 16384
                                    + (data[6] & 0xFF) * 128 + (data[7] & 0xFF);
                            SoftChannel[] channels = synth.channels;
                            for (int i = 0; i < channels.length; i++)
                                if ((channelmask & (1 << i)) != 0)
                                    channels[i].tuning = tuning;
                            break;
                        }
                        default:
                            break;
                        }
                        break;
                    case 0x09:  // General Midi Message
                        subid2 = data[4] & 0xFF;
                        switch (subid2) {
                        case 0x01:  // General Midi 1 On
                            synth.setGeneralMidiMode(1);
                            reset();
                            break;
                        case 0x02:  // General Midi Off
                            synth.setGeneralMidiMode(0);
                            reset();
```

```
                    break;
                case 0x03:  // General MidI Level 2 On
                    synth.setGeneralMidiMode(2);
                    reset();
                    break;
                default:
                    break;
                }
                break;
            case 0x0A: // DLS Message
                subid2 = data[4] & 0xFF;
                switch (subid2) {
                case 0x01:  // DLS On
                    if (synth.getGeneralMidiMode() == 0)
                        synth.setGeneralMidiMode(1);
                    synth.voice_allocation_mode = 1;
                    reset();
                    break;
                case 0x02:  // DLS Off
                    synth.setGeneralMidiMode(0);
                    synth.voice_allocation_mode = 0;
                    reset();
                    break;
                case 0x03:  // DLS Static Voice Allocation Off
                    synth.voice_allocation_mode = 0;
                    break;
                case 0x04:  // DLS Static Voice Allocation On
                    synth.voice_allocation_mode = 1;
                    break;
                default:
                    break;
                }
                break;

            default:
                break;
            }
        }
    }

    // Universal Real-Time SysEx
    if ((data[1] & 0xFF) == 0x7F) {
        int deviceID = data[2] & 0xFF;
        if (deviceID == 0x7F || deviceID == synth.getDeviceID()) {
            int subid1 = data[3] & 0xFF;
            int subid2;
            switch (subid1) {
            case 0x04: // Device Control

                subid2 = data[4] & 0xFF;
                switch (subid2) {
                case 0x01: // Master Volume
                case 0x02: // Master Balane
                case 0x03: // Master fine tuning
                case 0x04: // Master coarse tuning
                    int val = (data[5] & 0x7F)
                            + ((data[6] & 0x7F) * 128);
                    if (subid2 == 0x01)
                        setVolume(val);
                    else if (subid2 == 0x02)
                        setBalance(val);
                    else if (subid2 == 0x03)
```

```
                    setFineTuning(val);
                else if (subid2 == 0x04)
                    setCoarseTuning(val);
                break;
            case 0x05: // Global Parameter Control
                int ix = 5;
                int slotPathLen = (data[ix++] & 0xFF);
                int paramWidth = (data[ix++] & 0xFF);
                int valueWidth = (data[ix++] & 0xFF);
                int[] slotPath = new int[slotPathLen];
                for (int i = 0; i < slotPathLen; i++) {
                    int msb = (data[ix++] & 0xFF);
                    int lsb = (data[ix++] & 0xFF);
                    slotPath[i] = msb * 128 + lsb;
                }
                int paramCount = (data.length - 1 - ix)
                        / (paramWidth + valueWidth);
                long[] params = new long[paramCount];
                long[] values = new long[paramCount];
                for (int i = 0; i < paramCount; i++) {
                    values[i] = 0;
                    for (int j = 0; j < paramWidth; j++)
                        params[i] = params[i] * 128
                                + (data[ix++] & 0xFF);
                    for (int j = 0; j < valueWidth; j++)
                        values[i] = values[i] * 128
                                + (data[ix++] & 0xFF);

                }
                globalParameterControlChange(slotPath, params, values);
                break;
            default:
                break;
            }
            break;

        case 0x08:  // MIDI Tuning Standard
            subid2 = data[4] & 0xFF;
            switch (subid2) {
            case 0x02:  // SINGLE NOTE TUNING CHANGE (REAL-TIME)
            {
                // http://www.midi.org/about-midi/tuning.shtml
                SoftTuning tuning = synth.getTuning(new Patch(0,
                        data[5] & 0xFF));
                tuning.load(data);
                SoftVoice[] voices = synth.getVoices();
                for (int i = 0; i < voices.length; i++)
                    if (voices[i].active)
                        if (voices[i].tuning == tuning)
                            voices[i].updateTuning(tuning);
                break;
            }
            case 0x07:  // SINGLE NOTE TUNING CHANGE (REAL-TIME)
                        // (BANK)
            {
                // http://www.midi.org/about-midi/tuning_extens.shtml
                SoftTuning tuning = synth.getTuning(new Patch(
                        data[5] & 0xFF, data[6] & 0xFF));
                tuning.load(data);
                SoftVoice[] voices = synth.getVoices();
                for (int i = 0; i < voices.length; i++)
                    if (voices[i].active)
```

```
                        if (voices[i].tuning == tuning)
                            voices[i].updateTuning(tuning);
                    break;
                }
                case 0x08:  // scale/octave tuning 1-byte form
                            //(Real-Time)
                case 0x09:  // scale/octave tuning 2-byte form
                            // (Real-Time)
                {
                    // http://www.midi.org/about-midi/tuning-scale.shtml
                    SoftTuning tuning = new SoftTuning(data);
                    int channelmask = (data[5] & 0xFF) * 16384
                            + (data[6] & 0xFF) * 128 + (data[7] & 0xFF);
                    SoftChannel[] channels = synth.channels;
                    for (int i = 0; i < channels.length; i++)
                        if ((channelmask & (1 << i)) != 0)
                            channels[i].tuning = tuning;
                    SoftVoice[] voices = synth.getVoices();
                    for (int i = 0; i < voices.length; i++)
                        if (voices[i].active)
                            if ((channelmask & (1 << (voices[i].channel))) != 0)
                                voices[i].updateTuning(tuning);
                    break;
                }
                default:
                    break;
                }
                break;
            case 0x09:  // Control Destination Settings
                subid2 = data[4] & 0xFF;
                switch (subid2) {
                case 0x01: // Channel Pressure
                {
                    int[] destinations = new int[(data.length - 7) / 2];
                    int[] ranges = new int[(data.length - 7) / 2];
                    int ix = 0;
                    for (int j = 6; j < data.length - 1; j += 2) {
                        destinations[ix] = data[j] & 0xFF;
                        ranges[ix] = data[j + 1] & 0xFF;
                        ix++;
                    }
                    int channel = data[5] & 0xFF;
                    SoftChannel softchannel = synth.channels[channel];
                    softchannel.mapChannelPressureToDestination(
                            destinations, ranges);
                    break;
                }
                case 0x02: // Poly Pressure
                {
                    int[] destinations = new int[(data.length - 7) / 2];
                    int[] ranges = new int[(data.length - 7) / 2];
                    int ix = 0;
                    for (int j = 6; j < data.length - 1; j += 2) {
                        destinations[ix] = data[j] & 0xFF;
                        ranges[ix] = data[j + 1] & 0xFF;
                        ix++;
                    }
                    int channel = data[5] & 0xFF;
                    SoftChannel softchannel = synth.channels[channel];
                    softchannel.mapPolyPressureToDestination(
                            destinations, ranges);
                    break;
```

```
                        }
                        case 0x03: // Control Change
                        {
                            int[] destinations = new int[(data.length - 7) / 2];
                            int[] ranges = new int[(data.length - 7) / 2];
                            int ix = 0;
                            for (int j = 7; j < data.length - 1; j += 2) {
                                destinations[ix] = data[j] & 0xFF;
                                ranges[ix] = data[j + 1] & 0xFF;
                                ix++;
                            }
                            int channel = data[5] & 0xFF;
                            SoftChannel softchannel = synth.channels[channel];
                            int control = data[6] & 0xFF;
                            softchannel.mapControlToDestination(control,
                                    destinations, ranges);
                            break;
                        }
                        default:
                            break;
                        }
                        break;

                    case 0x0A:  // Key Based Instrument Control
                    {
                        subid2 = data[4] & 0xFF;
                        switch (subid2) {
                        case 0x01: // Basic Message
                            int channel = data[5] & 0xFF;
                            int keynumber = data[6] & 0xFF;
                            SoftChannel softchannel = synth.channels[channel];
                            for (int j = 7; j < data.length - 1; j += 2) {
                                int controlnumber = data[j] & 0xFF;
                                int controlvalue = data[j + 1] & 0xFF;
                                softchannel.controlChangePerNote(keynumber,
                                        controlnumber, controlvalue);
                            }
                            break;
                        default:
                            break;
                        }
                        break;
                    }
                    default:
                        break;
                    }
                }

            }
        }

        private void processMessages(long timeStamp) {
            Iterator<Entry<Long, Object>> iter = midimessages.entrySet().iterator();
            while (iter.hasNext()) {
                Entry<Long, Object> entry = iter.next();
                if (entry.getKey() >= (timeStamp + msec_buffer_len))
                    return;
                long msec_delay = entry.getKey() - timeStamp;
                delay_midievent = (int)(msec_delay * (samplerate / 1000000.0) + 0.5);
                if(delay_midievent > max_delay_midievent)
                    delay_midievent = max_delay_midievent;
```

```
            if(delay_midievent < 0)
                delay_midievent = 0;
            processMessage(entry.getValue());
            iter.remove();
        }
        delay_midievent = 0;
    }

    protected void processAudioBuffers() {

        if(synth.weakstream != null && synth.weakstream.silent_samples != 0)
        {
            sample_pos += synth.weakstream.silent_samples;
            synth.weakstream.silent_samples = 0;
        }

        for (int i = 0; i < buffers.length; i++) {
            if(i != CHANNEL_DELAY_LEFT &&
                    i != CHANNEL_DELAY_RIGHT &&
                    i != CHANNEL_DELAY_MONO &&
                    i != CHANNEL_DELAY_EFFECT1 &&
                    i != CHANNEL_DELAY_EFFECT2)
                buffers[i].clear();
        }

        if(!buffers[CHANNEL_DELAY_LEFT].isSilent())
        {
            buffers[CHANNEL_LEFT].swap(buffers[CHANNEL_DELAY_LEFT]);
        }
        if(!buffers[CHANNEL_DELAY_RIGHT].isSilent())
        {
            buffers[CHANNEL_RIGHT].swap(buffers[CHANNEL_DELAY_RIGHT]);
        }
        if(!buffers[CHANNEL_DELAY_MONO].isSilent())
        {
            buffers[CHANNEL_MONO].swap(buffers[CHANNEL_DELAY_MONO]);
        }
        if(!buffers[CHANNEL_DELAY_EFFECT1].isSilent())
        {
            buffers[CHANNEL_EFFECT1].swap(buffers[CHANNEL_DELAY_EFFECT1]);
        }
        if(!buffers[CHANNEL_DELAY_EFFECT2].isSilent())
        {
            buffers[CHANNEL_EFFECT2].swap(buffers[CHANNEL_DELAY_EFFECT2]);
        }

        double volume_left;
        double volume_right;

        SoftChannelMixerContainer[] act_registeredMixers;

        // perform control logic
        synchronized (control_mutex) {

            long msec_pos = (long)(sample_pos * (1000000.0 / samplerate));

            processMessages(msec_pos);

            if (active_sensing_on) {
                // Active Sensing
                // if no message occurs for max 1000 ms
                // then do AllSoundOff on all channels
```

```
            if ((msec_pos - msec_last_activity) > 1000000) {
                active_sensing_on = false;
                for (SoftChannel c : synth.channels)
                    c.allSoundOff();
            }

        }

        for (int i = 0; i < voicestatus.length; i++)
            if (voicestatus[i].active)
                voicestatus[i].processControlLogic();
        sample_pos += buffer_len;

        double volume = co_master_volume[0];
        volume_left = volume;
        volume_right = volume;

        double balance = co_master_balance[0];
        if (balance > 0.5)
            volume_left *= (1 - balance) * 2;
        else
            volume_right *= balance * 2;

        chorus.processControlLogic();
        reverb.processControlLogic();
        agc.processControlLogic();

        if (cur_registeredMixers == null) {
            if (registeredMixers != null) {
                cur_registeredMixers =
                        new SoftChannelMixerContainer[registeredMixers.size()];
                registeredMixers.toArray(cur_registeredMixers);
            }
        }

        act_registeredMixers = cur_registeredMixers;
        if (act_registeredMixers != null)
            if (act_registeredMixers.length == 0)
                act_registeredMixers = null;

    }

    if (act_registeredMixers != null) {

        // Make backup of left,right,mono channels
        SoftAudioBuffer leftbak = buffers[CHANNEL_LEFT];
        SoftAudioBuffer rightbak = buffers[CHANNEL_RIGHT];
        SoftAudioBuffer monobak = buffers[CHANNEL_MONO];
        SoftAudioBuffer delayleftbak = buffers[CHANNEL_DELAY_LEFT];
        SoftAudioBuffer delayrightbak = buffers[CHANNEL_DELAY_RIGHT];
        SoftAudioBuffer delaymonobak = buffers[CHANNEL_DELAY_MONO];

        int bufferlen = buffers[CHANNEL_LEFT].getSize();

        float[][] cbuffer = new float[nrofchannels][];
        float[][] obuffer = new float[nrofchannels][];
        obuffer[0] = leftbak.array();
        if (nrofchannels != 1)
            obuffer[1] = rightbak.array();

        for (SoftChannelMixerContainer cmixer : act_registeredMixers) {

```

```
                // Reroute default left,right output
                // to channelmixer left,right input/output
                buffers[CHANNEL_LEFT] =  cmixer.buffers[CHANNEL_LEFT];
                buffers[CHANNEL_RIGHT] = cmixer.buffers[CHANNEL_RIGHT];
                buffers[CHANNEL_MONO] = cmixer.buffers[CHANNEL_MONO];
                buffers[CHANNEL_DELAY_LEFT] = cmixer.buffers[CHANNEL_DELAY_LEFT];
                buffers[CHANNEL_DELAY_RIGHT] = cmixer.buffers[CHANNEL_DELAY_RIGHT];
                buffers[CHANNEL_DELAY_MONO] = cmixer.buffers[CHANNEL_DELAY_MONO];

                buffers[CHANNEL_LEFT].clear();
                buffers[CHANNEL_RIGHT].clear();
                buffers[CHANNEL_MONO].clear();

                if(!buffers[CHANNEL_DELAY_LEFT].isSilent())
                {
                    buffers[CHANNEL_LEFT].swap(buffers[CHANNEL_DELAY_LEFT]);
                }
                if(!buffers[CHANNEL_DELAY_RIGHT].isSilent())
                {
                    buffers[CHANNEL_RIGHT].swap(buffers[CHANNEL_DELAY_RIGHT]);
                }
                if(!buffers[CHANNEL_DELAY_MONO].isSilent())
                {
                    buffers[CHANNEL_MONO].swap(buffers[CHANNEL_DELAY_MONO]);
                }

                cbuffer[0] = buffers[CHANNEL_LEFT].array();
                if (nrofchannels != 1)
                    cbuffer[1] = buffers[CHANNEL_RIGHT].array();

                boolean hasactivevoices = false;
                for (int i = 0; i < voicestatus.length; i++)
                    if (voicestatus[i].active)
                        if (voicestatus[i].channelmixer == cmixer.mixer) {
                            voicestatus[i].processAudioLogic(buffers);
                            hasactivevoices = true;
                        }


                if(!buffers[CHANNEL_MONO].isSilent())
                {
                    float[] mono = buffers[CHANNEL_MONO].array();
                    float[] left = buffers[CHANNEL_LEFT].array();
                    if (nrofchannels != 1) {
                        float[] right = buffers[CHANNEL_RIGHT].array();
                        for (int i = 0; i < bufferlen; i++) {
                            float v = mono[i];
                            left[i] += v;
                            right[i] += v;
                        }
                    }
                    else
                    {
                        for (int i = 0; i < bufferlen; i++) {
                            left[i] += mono[i];
                        }
                    }
                }

                if (!cmixer.mixer.process(cbuffer, 0, bufferlen)) {
                    synchronized (control_mutex) {
                        registeredMixers.remove(cmixer);
```

```
                    cur_registeredMixers = null;
                }
            }

            for (int i = 0; i < cbuffer.length; i++) {
                float[] cbuff = cbuffer[i];
                float[] obuff = obuffer[i];
                for (int j = 0; j < bufferlen; j++)
                    obuff[j] += cbuff[j];
            }

            if (!hasactivevoices) {
                synchronized (control_mutex) {
                    if (stoppedMixers != null) {
                        if (stoppedMixers.contains(cmixer)) {
                            stoppedMixers.remove(cmixer);
                            cmixer.mixer.stop();
                        }
                    }
                }
            }

        }

        buffers[CHANNEL_LEFT] = leftbak;
        buffers[CHANNEL_RIGHT] = rightbak;
        buffers[CHANNEL_MONO] = monobak;
        buffers[CHANNEL_DELAY_LEFT] = delayleftbak;
        buffers[CHANNEL_DELAY_RIGHT] = delayrightbak;
        buffers[CHANNEL_DELAY_MONO] = delaymonobak;

    }

    for (int i = 0; i < voicestatus.length; i++)
        if (voicestatus[i].active)
            if (voicestatus[i].channelmixer == null)
                voicestatus[i].processAudioLogic(buffers);

    if(!buffers[CHANNEL_MONO].isSilent())
    {
        float[] mono = buffers[CHANNEL_MONO].array();
        float[] left = buffers[CHANNEL_LEFT].array();
        int bufferlen = buffers[CHANNEL_LEFT].getSize();
        if (nrofchannels != 1) {
            float[] right = buffers[CHANNEL_RIGHT].array();
            for (int i = 0; i < bufferlen; i++) {
                float v = mono[i];
                left[i] += v;
                right[i] += v;
            }
        }
        else
        {
            for (int i = 0; i < bufferlen; i++) {
                left[i] += mono[i];
            }
        }
    }

    // Run effects
    if (synth.chorus_on)
        chorus.processAudio();
```

```
681
682         if (synth.reverb_on)
683             reverb.processAudio();
684
685         if (nrofchannels == 1)
686             volume_left = (volume_left + volume_right) / 2;
687
688         // Set Volume / Balance
689         if (last_volume_left != volume_left || last_volume_right != volume_right) {
690             float[] left = buffers[CHANNEL_LEFT].array();
691             float[] right = buffers[CHANNEL_RIGHT].array();
692             int bufferlen = buffers[CHANNEL_LEFT].getSize();
693
694             float amp;
695             float amp_delta;
696             amp = (float)(last_volume_left * last_volume_left);
697             amp_delta = (float)((volume_left * volume_left - amp) / bufferlen);
698             for (int i = 0; i < bufferlen; i++) {
699                 amp += amp_delta;
700                 left[i] *= amp;
701             }
702             if (nrofchannels != 1) {
703                 amp = (float)(last_volume_right * last_volume_right);
704                 amp_delta = (float)((volume_right*volume_right - amp) / bufferlen);
705                 for (int i = 0; i < bufferlen; i++) {
706                     amp += amp_delta;
707                     right[i] *= volume_right;
708                 }
709             }
710             last_volume_left = volume_left;
711             last_volume_right = volume_right;
712
713         } else {
714             if (volume_left != 1.0 || volume_right != 1.0) {
715                 float[] left = buffers[CHANNEL_LEFT].array();
716                 float[] right = buffers[CHANNEL_RIGHT].array();
717                 int bufferlen = buffers[CHANNEL_LEFT].getSize();
718                 float amp;
719                 amp = (float) (volume_left * volume_left);
720                 for (int i = 0; i < bufferlen; i++)
721                     left[i] *= amp;
722                 if (nrofchannels != 1) {
723                     amp = (float)(volume_right * volume_right);
724                     for (int i = 0; i < bufferlen; i++)
725                         right[i] *= amp;
726                 }
727
728             }
729         }
730
731         if(buffers[CHANNEL_LEFT].isSilent()
732             && buffers[CHANNEL_RIGHT].isSilent())
733         {
734
735             int midimessages_size;
736             synchronized (control_mutex) {
737                 midimessages_size = midimessages.size();
738             }
739
740             if(midimessages_size == 0)
741             {
742                 pusher_silent_count++;
```

```
                if(pusher_silent_count > 5)
                {
                    pusher_silent_count = 0;
                    synchronized (control_mutex) {
                        pusher_silent = true;
                        if(synth.weakstream != null)
                            synth.weakstream.setInputStream(null);
                    }
                }
            }
        }
        else
            pusher_silent_count = 0;

        if (synth.agc_on)
            agc.processAudio();

    }

    // Must only we called within control_mutex synchronization
    public void activity()
    {
        long silent_samples = 0;
        if(pusher_silent)
        {
            pusher_silent = false;
            if(synth.weakstream != null)
            {
                synth.weakstream.setInputStream(ais);
                silent_samples = synth.weakstream.silent_samples;;
            }
        }
        msec_last_activity = (long)((sample_pos + silent_samples)
                * (1000000.0 / samplerate));
    }

    public void stopMixer(ModelChannelMixer mixer) {
        if (stoppedMixers == null)
            stoppedMixers = new HashSet<ModelChannelMixer>();
        stoppedMixers.add(mixer);
    }

    public void registerMixer(ModelChannelMixer mixer) {
        if (registeredMixers == null)
            registeredMixers = new HashSet<SoftChannelMixerContainer>();
        SoftChannelMixerContainer mixercontainer = new SoftChannelMixerContainer();
        mixercontainer.buffers = new SoftAudioBuffer[6];
        for (int i = 0; i < mixercontainer.buffers.length; i++) {
            mixercontainer.buffers[i] =
                new SoftAudioBuffer(buffer_len, synth.getFormat());
        }
        mixercontainer.mixer = mixer;
        registeredMixers.add(mixercontainer);
        cur_registeredMixers = null;
    }

    public SoftMainMixer(SoftSynthesizer synth) {
        this.synth = synth;

        sample_pos = 0;

        co_master_balance[0] = 0.5;
```

```
        co_master_volume[0] = 1;
        co_master_coarse_tuning[0] = 0.5;
        co_master_fine_tuning[0] = 0.5;

        msec_buffer_len = (long) (1000000.0 / synth.getControlRate());
        samplerate = synth.getFormat().getSampleRate();
        nrofchannels = synth.getFormat().getChannels();

        int buffersize = (int) (synth.getFormat().getSampleRate()
                                / synth.getControlRate());

        buffer_len = buffersize;

        max_delay_midievent = buffersize;

        control_mutex = synth.control_mutex;
        buffers = new SoftAudioBuffer[14];
        for (int i = 0; i < buffers.length; i++) {
            buffers[i] = new SoftAudioBuffer(buffersize, synth.getFormat());
        }
        voicestatus = synth.getVoices();

        reverb = new SoftReverb();
        chorus = new SoftChorus();
        agc = new SoftLimiter();

        float samplerate = synth.getFormat().getSampleRate();
        float controlrate = synth.getControlRate();
        reverb.init(samplerate, controlrate);
        chorus.init(samplerate, controlrate);
        agc.init(samplerate, controlrate);

        reverb.setLightMode(synth.reverb_light);

        reverb.setMixMode(true);
        chorus.setMixMode(true);
        agc.setMixMode(false);

        chorus.setInput(0, buffers[CHANNEL_EFFECT2]);
        chorus.setOutput(0, buffers[CHANNEL_LEFT]);
        if (nrofchannels != 1)
            chorus.setOutput(1, buffers[CHANNEL_RIGHT]);
        chorus.setOutput(2, buffers[CHANNEL_EFFECT1]);

        reverb.setInput(0, buffers[CHANNEL_EFFECT1]);
        reverb.setOutput(0, buffers[CHANNEL_LEFT]);
        if (nrofchannels != 1)
            reverb.setOutput(1, buffers[CHANNEL_RIGHT]);

        agc.setInput(0, buffers[CHANNEL_LEFT]);
        if (nrofchannels != 1)
            agc.setInput(1, buffers[CHANNEL_RIGHT]);
        agc.setOutput(0, buffers[CHANNEL_LEFT]);
        if (nrofchannels != 1)
            agc.setOutput(1, buffers[CHANNEL_RIGHT]);

        InputStream in = new InputStream() {

            private SoftAudioBuffer[] buffers = SoftMainMixer.this.buffers;
            private int nrofchannels
                    = SoftMainMixer.this.synth.getFormat().getChannels();
            private int buffersize = buffers[0].getSize();
```

```
            private byte[] bbuffer = new byte[buffersize
                    * (SoftMainMixer.this.synth.getFormat()
                        .getSampleSizeInBits() / 8)
                    * nrofchannels];
            private int bbuffer_pos = 0;
            private byte[] single = new byte[1];

            public void fillBuffer() {
                /*
                boolean pusher_silent2;
                synchronized (control_mutex) {
                    pusher_silent2 = pusher_silent;
                }
                if(!pusher_silent2)*/
                processAudioBuffers();
                for (int i = 0; i < nrofchannels; i++)
                    buffers[i].get(bbuffer, i);
                bbuffer_pos = 0;
            }

            public int read(byte[] b, int off, int len) {
                int bbuffer_len = bbuffer.length;
                int offlen = off + len;
                int orgoff = off;
                byte[] bbuffer = this.bbuffer;
                while (off < offlen) {
                    if (available() == 0)
                        fillBuffer();
                    else {
                        int bbuffer_pos = this.bbuffer_pos;
                        while (off < offlen && bbuffer_pos < bbuffer_len)
                            b[off++] = bbuffer[bbuffer_pos++];
                        this.bbuffer_pos = bbuffer_pos;
                        if (!readfully)
                            return off - orgoff;
                    }
                }
                return len;
            }

            public int read() throws IOException {
                int ret = read(single);
                if (ret == -1)
                    return -1;
                return single[0] & 0xFF;
            }

            public int available() {
                return bbuffer.length - bbuffer_pos;
            }

            public void close() {
                SoftMainMixer.this.synth.close();
            }
        };

        ais = new AudioInputStream(in, synth.getFormat(), AudioSystem.NOT_SPECIFIED);

    }

    public AudioInputStream getInputStream() {
        return ais;
```

```
    }

    public void reset() {

        SoftChannel[] channels = synth.channels;
        for (int i = 0; i < channels.length; i++) {
            channels[i].allSoundOff();
            channels[i].resetAllControllers(true);

            if (synth.getGeneralMidiMode() == 2) {
                if (i == 9)
                    channels[i].programChange(0, 0x78 * 128);
                else
                    channels[i].programChange(0, 0x79 * 128);
            } else
                channels[i].programChange(0, 0);
        }
        setVolume(0x7F * 128 + 0x7F);
        setBalance(0x40 * 128 + 0x00);
        setCoarseTuning(0x40 * 128 + 0x00);
        setFineTuning(0x40 * 128 + 0x00);
        // Reset Reverb
        globalParameterControlChange(
                new int[]{0x01 * 128 + 0x01}, new long[]{0}, new long[]{4});
        // Reset Chorus
        globalParameterControlChange(
                new int[]{0x01 * 128 + 0x02}, new long[]{0}, new long[]{2});
    }

    public void setVolume(int value) {
        synchronized (control_mutex) {
            co_master_volume[0] = value / 16384.0;
        }
    }

    public void setBalance(int value) {
        synchronized (control_mutex) {
            co_master_balance[0] = value / 16384.0;
        }
    }

    public void setFineTuning(int value) {
        synchronized (control_mutex) {
            co_master_fine_tuning[0] = value / 16384.0;
        }
    }

    public void setCoarseTuning(int value) {
        synchronized (control_mutex) {
            co_master_coarse_tuning[0] = value / 16384.0;
        }
    }

    public int getVolume() {
        synchronized (control_mutex) {
            return (int) (co_master_volume[0] * 16384.0);
        }
    }

    public int getBalance() {
        synchronized (control_mutex) {
            return (int) (co_master_balance[0] * 16384.0);
```

```
        }
    }

    public int getFineTuning() {
        synchronized (control_mutex) {
            return (int) (co_master_fine_tuning[0] * 16384.0);
        }
    }

    public int getCoarseTuning() {
        synchronized (control_mutex) {
            return (int) (co_master_coarse_tuning[0] * 16384.0);
        }
    }

    public void globalParameterControlChange(int[] slothpath, long[] params,
            long[] paramsvalue) {
        if (slothpath.length == 0)
            return;

        synchronized (control_mutex) {

            // slothpath: 01xx are reserved only for GM2

            if (slothpath[0] == 0x01 * 128 + 0x01) {
                for (int i = 0; i < paramsvalue.length; i++) {
                    reverb.globalParameterControlChange(slothpath, params[i],
                            paramsvalue[i]);
                }
            }
            if (slothpath[0] == 0x01 * 128 + 0x02) {
                for (int i = 0; i < paramsvalue.length; i++) {
                    chorus.globalParameterControlChange(slothpath, params[i],
                            paramsvalue[i]);
                }

            }

        }
    }

    public void processMessage(Object object) {
        if (object instanceof byte[])
            processMessage((byte[]) object);
        if (object instanceof MidiMessage)
            processMessage((MidiMessage)object);
    }

    public void processMessage(MidiMessage message) {
        if (message instanceof ShortMessage) {
            ShortMessage sms = (ShortMessage)message;
            processMessage(sms.getChannel(), sms.getCommand(),
                    sms.getData1(), sms.getData2());
            return;
        }
        processMessage(message.getMessage());
    }

    public void processMessage(byte[] data) {
        int status = 0;
        if (data.length > 0)
            status = data[0] & 0xFF;
```

```
        if (status == 0xF0) {
            processSystemExclusiveMessage(data);
            return;
        }

        int cmd = (status & 0xF0);
        int ch = (status & 0x0F);

        int data1;
        int data2;
        if (data.length > 1)
            data1 = data[1] & 0xFF;
        else
            data1 = 0;
        if (data.length > 2)
            data2 = data[2] & 0xFF;
        else
            data2 = 0;

        processMessage(ch, cmd, data1, data2);

    }

    public void processMessage(int ch, int cmd, int data1, int data2) {
        synchronized (synth.control_mutex) {
            activity();
        }

        if (cmd == 0xF0) {
            int status = cmd | ch;
            switch (status) {
            case ShortMessage.ACTIVE_SENSING:
                synchronized (synth.control_mutex) {
                    active_sensing_on = true;
                }
                break;
            default:
                break;
            }
            return;
        }

        SoftChannel[] channels = synth.channels;
        if (ch >= channels.length)
            return;
        SoftChannel softchannel = channels[ch];

        switch (cmd) {
        case ShortMessage.NOTE_ON:
            if(delay_midievent != 0)
                softchannel.noteOn(data1, data2, delay_midievent);
            else
                softchannel.noteOn(data1, data2);
            break;
        case ShortMessage.NOTE_OFF:
            softchannel.noteOff(data1, data2);
            break;
        case ShortMessage.POLY_PRESSURE:
            softchannel.setPolyPressure(data1, data2);
            break;
        case ShortMessage.CONTROL_CHANGE:
```

```
            softchannel.controlChange(data1, data2);
            break;
        case ShortMessage.PROGRAM_CHANGE:
            softchannel.programChange(data1);
            break;
        case ShortMessage.CHANNEL_PRESSURE:
            softchannel.setChannelPressure(data1);
            break;
        case ShortMessage.PITCH_BEND:
            softchannel.setPitchBend(data1 + data2 * 128);
            break;
        default:
            break;
        }

    }

    public long getMicrosecondPosition() {
        if(pusher_silent)
        {
            if(synth.weakstream != null)
            {
                return (long)((sample_pos  + synth.weakstream.silent_samples)
                        * (1000000.0 / samplerate));
            }
        }
        return (long)(sample_pos * (1000000.0 / samplerate));
    }

    public void close() {
    }
}
```

96 com/sun/media/sound/SoftMidiAudioFileReader.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

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

import javax.sound.midi.InvalidMidiDataException;
import javax.sound.midi.MetaMessage;
import javax.sound.midi.MidiEvent;
import javax.sound.midi.MidiMessage;
import javax.sound.midi.MidiSystem;
import javax.sound.midi.MidiUnavailableException;
import javax.sound.midi.Receiver;
import javax.sound.midi.Sequence;
import javax.sound.midi.Track;
import javax.sound.sampled.AudioFileFormat;
import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.UnsupportedAudioFileException;
import javax.sound.sampled.AudioFileFormat.Type;
import javax.sound.sampled.spi.AudioFileReader;

/**
 * MIDI File Audio Renderer/Reader
 *
 * @author Karl Helgason
 */
public class SoftMidiAudioFileReader extends AudioFileReader {

    public static final Type MIDI = new Type("MIDI", "mid");
    private static AudioFormat format = new AudioFormat(44100, 16, 2, true, false);

    public AudioFileFormat getAudioFileFormat(Sequence seq)
            throws UnsupportedAudioFileException, IOException {

```

```
        long totallen = seq.getMicrosecondLength() / 1000000;
        long len = (long) (format.getFrameRate() * (totallen + 4));
        return new AudioFileFormat(MIDI, format, (int) len);
    }

    public AudioInputStream getAudioInputStream(Sequence seq)
            throws UnsupportedAudioFileException, IOException {
        AudioSynthesizer synth = (AudioSynthesizer) new SoftSynthesizer();
        AudioInputStream stream;
        Receiver recv;
        try {
            stream = synth.openStream(format, null);
            recv = synth.getReceiver();
        } catch (MidiUnavailableException e) {
            throw new IOException(e.toString());
        }
        float divtype = seq.getDivisionType();
        Track[] tracks = seq.getTracks();
        int[] trackspos = new int[tracks.length];
        int mpq = 500000;
        int seqres = seq.getResolution();
        long lasttick = 0;
        long curtime = 0;
        while (true) {
            MidiEvent selevent = null;
            int seltrack = -1;
            for (int i = 0; i < tracks.length; i++) {
                int trackpos = trackspos[i];
                Track track = tracks[i];
                if (trackpos < track.size()) {
                    MidiEvent event = track.get(trackpos);
                    if (selevent == null || event.getTick() < selevent.getTick()) {
                        selevent = event;
                        seltrack = i;
                    }
                }
            }
            if (seltrack == -1)
                break;
            trackspos[seltrack]++;
            long tick = selevent.getTick();
            if (divtype == Sequence.PPQ)
                curtime += ((tick - lasttick) * mpq) / seqres;
            else
                curtime = (long) ((tick * 1000000.0 * divtype) / seqres);
            lasttick = tick;
            MidiMessage msg = selevent.getMessage();
            if (msg instanceof MetaMessage) {
                if (divtype == Sequence.PPQ) {
                    if (((MetaMessage) msg).getType() == 0x51) {
                        byte[] data = ((MetaMessage) msg).getData();
                        mpq = ((data[0] & 0xff) << 16)
                                | ((data[1] & 0xff) << 8) | (data[2] & 0xff);
                    }
                }
            } else {
                recv.send(msg, curtime);
            }
        }

        long totallen = curtime / 1000000;
        long len = (long) (stream.getFormat().getFrameRate() * (totallen + 4));
```

```
        stream = new AudioInputStream(stream, stream.getFormat(), len);
        return stream;
    }

    public AudioInputStream getAudioInputStream(InputStream inputstream)
            throws UnsupportedAudioFileException, IOException {

        inputstream.mark(200);
        Sequence seq;
        try {
            seq = MidiSystem.getSequence(inputstream);
        } catch (InvalidMidiDataException e) {
            inputstream.reset();
            throw new UnsupportedAudioFileException();
        } catch (IOException e) {
            inputstream.reset();
            throw new UnsupportedAudioFileException();
        }
        return getAudioInputStream(seq);
    }

    public AudioFileFormat getAudioFileFormat(URL url)
            throws UnsupportedAudioFileException, IOException {
        Sequence seq;
        try {
            seq = MidiSystem.getSequence(url);
        } catch (InvalidMidiDataException e) {
            throw new UnsupportedAudioFileException();
        } catch (IOException e) {
            throw new UnsupportedAudioFileException();
        }
        return getAudioFileFormat(seq);
    }

    public AudioFileFormat getAudioFileFormat(File file)
            throws UnsupportedAudioFileException, IOException {
        Sequence seq;
        try {
            seq = MidiSystem.getSequence(file);
        } catch (InvalidMidiDataException e) {
            throw new UnsupportedAudioFileException();
        } catch (IOException e) {
            throw new UnsupportedAudioFileException();
        }
        return getAudioFileFormat(seq);
    }

    public AudioInputStream getAudioInputStream(URL url)
            throws UnsupportedAudioFileException, IOException {
        Sequence seq;
        try {
            seq = MidiSystem.getSequence(url);
        } catch (InvalidMidiDataException e) {
            throw new UnsupportedAudioFileException();
        } catch (IOException e) {
            throw new UnsupportedAudioFileException();
        }
        return getAudioInputStream(seq);
    }

    public AudioInputStream getAudioInputStream(File file)
            throws UnsupportedAudioFileException, IOException {
```

```
        if (!file.getName().toLowerCase().endsWith(".mid"))
            throw new UnsupportedAudioFileException();
        Sequence seq;
        try {
            seq = MidiSystem.getSequence(file);
        } catch (InvalidMidiDataException e) {
            throw new UnsupportedAudioFileException();
        } catch (IOException e) {
            throw new UnsupportedAudioFileException();
        }
        return getAudioInputStream(seq);
    }

    public AudioFileFormat getAudioFileFormat(InputStream inputstream)
            throws UnsupportedAudioFileException, IOException {

        inputstream.mark(200);
        Sequence seq;
        try {
            seq = MidiSystem.getSequence(inputstream);
        } catch (InvalidMidiDataException e) {
            inputstream.reset();
            throw new UnsupportedAudioFileException();
        } catch (IOException e) {
            inputstream.reset();
            throw new UnsupportedAudioFileException();
        }
        return getAudioFileFormat(seq);
    }
}
```

97 com/sun/media/sound/SoftMixingClip.java

```
/*
 * Copyright 2008 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.Arrays;

import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.Clip;
import javax.sound.sampled.DataLine;
import javax.sound.sampled.LineEvent;
import javax.sound.sampled.LineUnavailableException;

/**
 * Clip implemention for the SoftMixingMixer.
 *
 * @author Karl Helgason
 */
public class SoftMixingClip extends SoftMixingDataLine implements Clip {

    private AudioFormat format;

    private int framesize;

    private byte[] data;

    private InputStream datastream = new InputStream() {

        public int read() throws IOException {
            byte[] b = new byte[1];
            int ret = read(b);
            if (ret < 0)
                return ret;
            return b[0] & 0xFF;
```

```
61          }
62
63          public int read(byte[] b, int off, int len) throws IOException {
64
65              if (_loopcount != 0) {
66                  int bloopend = _loopend * framesize;
67                  int bloopstart = _loopstart * framesize;
68                  int pos = _frameposition * framesize;
69
70                  if (pos + len >= bloopend)
71                      if (pos < bloopend) {
72                          int offend = off + len;
73                          int o = off;
74                          while (off != offend) {
75                              if (pos == bloopend) {
76                                  if (_loopcount == 0)
77                                      break;
78                                  pos = bloopstart;
79                                  if (_loopcount != LOOP_CONTINUOUSLY)
80                                      _loopcount--;
81                              }
82                              len = offend - off;
83                              int left = bloopend - pos;
84                              if (len > left)
85                                  len = left;
86                              System.arraycopy(data, pos, b, off, len);
87                              off += len;
88                          }
89                          if (_loopcount == 0) {
90                              len = offend - off;
91                              int left = bloopend - pos;
92                              if (len > left)
93                                  len = left;
94                              System.arraycopy(data, pos, b, off, len);
95                              off += len;
96                          }
97                          _frameposition = pos / framesize;
98                          return o - off;
99                      }
100             }
101
102             int pos = _frameposition * framesize;
103             int left = bufferSize - pos;
104             if (left == 0)
105                 return -1;
106             if (len > left)
107                 len = left;
108             System.arraycopy(data, pos, b, off, len);
109             _frameposition += len / framesize;
110             return len;
111         }
112
113     };
114
115     private int offset;
116
117     private int bufferSize;
118
119     private float[] readbuffer;
120
121     private boolean open = false;
122
```

```
123     private AudioFormat outputformat;
124
125     private int out_nrofchannels;
126
127     private int in_nrofchannels;
128
129     private int frameposition = 0;
130
131     private boolean frameposition_sg = false;
132
133     private boolean active_sg = false;
134
135     private int loopstart = 0;
136
137     private int loopend = -1;
138
139     private boolean active = false;
140
141     private int loopcount = 0;
142
143     private boolean _active = false;
144
145     private int _frameposition = 0;
146
147     private boolean loop_sg = false;
148
149     private int _loopcount = 0;
150
151     private int _loopstart = 0;
152
153     private int _loopend = -1;
154
155     private float _rightgain;
156
157     private float _leftgain;
158
159     private float _eff1gain;
160
161     private float _eff2gain;
162
163     private AudioFloatInputStream afis;
164
165     protected SoftMixingClip(SoftMixingMixer mixer, DataLine.Info info) {
166         super(mixer, info);
167     }
168
169     protected void processControlLogic() {
170
171         _rightgain = rightgain;
172         _leftgain = leftgain;
173         _eff1gain = eff1gain;
174         _eff2gain = eff2gain;
175
176         if (active_sg) {
177             _active = active;
178             active_sg = false;
179         } else {
180             active = _active;
181         }
182
183         if (frameposition_sg) {
184             _frameposition = frameposition;
```

```
            frameposition_sg = false;
            afis = null;
        } else {
            frameposition = _frameposition;
        }
        if (loop_sg) {
            _loopcount = loopcount;
            _loopstart = loopstart;
            _loopend = loopend;
        }

        if (afis == null) {
            afis = AudioFloatInputStream.getInputStream(new AudioInputStream(
                    datastream, format, AudioSystem.NOT_SPECIFIED));

            if (Math.abs(format.getSampleRate() - outputformat.getSampleRate()) > 0.000001)
                afis = new AudioFloatInputStreamResampler(afis, outputformat);
        }

    }

    protected void processAudioLogic(SoftAudioBuffer[] buffers) {
        if (_active) {
            float[] left = buffers[SoftMixingMainMixer.CHANNEL_LEFT].array();
            float[] right = buffers[SoftMixingMainMixer.CHANNEL_RIGHT].array();
            int bufferlen = buffers[SoftMixingMainMixer.CHANNEL_LEFT].getSize();

            int readlen = bufferlen * in_nrofchannels;
            if (readbuffer == null || readbuffer.length < readlen) {
                readbuffer = new float[readlen];
            }
            int ret = 0;
            try {
                ret = afis.read(readbuffer);
                if (ret == -1) {
                    _active = false;
                    return;
                }
                if (ret != in_nrofchannels)
                    Arrays.fill(readbuffer, ret, readlen, 0);
            } catch (IOException e) {
            }

            int in_c = in_nrofchannels;
            for (int i = 0, ix = 0; i < bufferlen; i++, ix += in_c) {
                left[i] += readbuffer[ix] * _leftgain;
            }

            if (out_nrofchannels != 1) {
                if (in_nrofchannels == 1) {
                    for (int i = 0, ix = 0; i < bufferlen; i++, ix += in_c) {
                        right[i] += readbuffer[ix] * _rightgain;
                    }
                } else {
                    for (int i = 0, ix = 1; i < bufferlen; i++, ix += in_c) {
                        right[i] += readbuffer[ix] * _rightgain;
                    }
                }

            }

            if (_eff1gain > 0.0002) {
```

```
247
248                 float[] eff1 = buffers[SoftMixingMainMixer.CHANNEL_EFFECT1]
249                         .array();
250                 for (int i = 0, ix = 0; i < bufferlen; i++, ix += in_c) {
251                     eff1[i] += readbuffer[ix] * _eff1gain;
252                 }
253                 if (in_nrofchannels == 2) {
254                     for (int i = 0, ix = 1; i < bufferlen; i++, ix += in_c) {
255                         eff1[i] += readbuffer[ix] * _eff1gain;
256                     }
257                 }
258             }
259
260             if (_eff2gain > 0.0002) {
261                 float[] eff2 = buffers[SoftMixingMainMixer.CHANNEL_EFFECT2]
262                         .array();
263                 for (int i = 0, ix = 0; i < bufferlen; i++, ix += in_c) {
264                     eff2[i] += readbuffer[ix] * _eff2gain;
265                 }
266                 if (in_nrofchannels == 2) {
267                     for (int i = 0, ix = 1; i < bufferlen; i++, ix += in_c) {
268                         eff2[i] += readbuffer[ix] * _eff2gain;
269                     }
270                 }
271             }
272
273         }
274     }
275
276     public int getFrameLength() {
277         return bufferSize / format.getFrameSize();
278     }
279
280     public long getMicrosecondLength() {
281         return (long) (getFrameLength() * (1000000.0 / (double) getFormat()
282                 .getSampleRate()));
283     }
284
285     public void loop(int count) {
286         LineEvent event = null;
287
288         synchronized (control_mutex) {
289             if (isOpen()) {
290                 if (active)
291                     return;
292                 active = true;
293                 active_sg = true;
294                 loopcount = count;
295                 event = new LineEvent(this, LineEvent.Type.START,
296                         getLongFramePosition());
297             }
298         }
299
300         if (event != null)
301             sendEvent(event);
302
303     }
304
305     public void open(AudioInputStream stream) throws LineUnavailableException,
306             IOException {
307         if (isOpen()) {
308             throw new IllegalStateException("Clip␣is␣already␣open␣with␣format␣"
```

```
                + getFormat() + " and frame lengh of " + getFrameLength());
        }
        if (AudioFloatConverter.getConverter(stream.getFormat()) == null)
            throw new IllegalArgumentException("Invalid format : "
                    + stream.getFormat().toString());

        if (stream.getFrameLength() != AudioSystem.NOT_SPECIFIED) {
            byte[] data = new byte[(int) stream.getFrameLength()
                    * stream.getFormat().getFrameSize()];
            int readsize = 512 * stream.getFormat().getFrameSize();
            int len = 0;
            while (len != data.length) {
                if (readsize > data.length - len)
                    readsize = data.length - len;
                int ret = stream.read(data, len, readsize);
                if (ret == -1)
                    break;
                if (ret == 0)
                    Thread.yield();
                len += ret;
            }
            open(stream.getFormat(), data, 0, len);
        } else {
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            byte[] b = new byte[512 * stream.getFormat().getFrameSize()];
            int r = 0;
            while ((r = stream.read(b)) != -1) {
                if (r == 0)
                    Thread.yield();
                baos.write(b, 0, r);
            }
            open(stream.getFormat(), baos.toByteArray(), 0, baos.size());
        }

    }

    public void open(AudioFormat format, byte[] data, int offset, int bufferSize)
            throws LineUnavailableException {
        synchronized (control_mutex) {
            if (isOpen()) {
                throw new IllegalStateException(
                        "Clip is already open with format " + getFormat()
                                + " and frame lengh of " + getFrameLength());
            }
            if (AudioFloatConverter.getConverter(format) == null)
                throw new IllegalArgumentException("Invalid format : "
                        + format.toString());
            if (bufferSize % format.getFrameSize() != 0)
                throw new IllegalArgumentException(
                        "Buffer size does not represent an integral number of sample frames!");

            this.data = data;
            this.offset = offset;
            this.bufferSize = bufferSize;
            this.format = format;
            this.framesize = format.getFrameSize();

            loopstart = 0;
            loopend = -1;
            loop_sg = true;

            if (!mixer.isOpen()) {
```

```
                mixer.open();
                mixer.implicitOpen = true;
            }

            outputformat = mixer.getFormat();
            out_nrofchannels = outputformat.getChannels();
            in_nrofchannels = format.getChannels();

            open = true;

            mixer.getMainMixer().openLine(this);
        }

    }

    public void setFramePosition(int frames) {
        synchronized (control_mutex) {
            frameposition_sg = true;
            frameposition = frames;
        }
    }

    public void setLoopPoints(int start, int end) {
        synchronized (control_mutex) {
            if (end != -1) {
                if (end < start)
                    throw new IllegalArgumentException("Invalid loop points : "
                            + start + " - " + end);
                if (end * framesize > bufferSize)
                    throw new IllegalArgumentException("Invalid loop points : "
                            + start + " - " + end);
            }
            if (start * framesize > bufferSize)
                throw new IllegalArgumentException("Invalid loop points : "
                        + start + " - " + end);
            if (0 < start)
                throw new IllegalArgumentException("Invalid loop points : "
                        + start + " - " + end);
            loopstart = start;
            loopend = end;
            loop_sg = true;
        }
    }

    public void setMicrosecondPosition(long microseconds) {
        setFramePosition((int) (microseconds * (((double) getFormat()
                .getSampleRate()) / 1000000.0)));
    }

    public int available() {
        return 0;
    }

    public void drain() {
    }

    public void flush() {
    }

    public int getBufferSize() {
        return bufferSize;
    }
```

```
433
434     public AudioFormat getFormat() {
435         return format;
436     }
437
438     public int getFramePosition() {
439         synchronized (control_mutex) {
440             return frameposition;
441         }
442     }
443
444     public float getLevel() {
445         return AudioSystem.NOT_SPECIFIED;
446     }
447
448     public long getLongFramePosition() {
449         return getFramePosition();
450     }
451
452     public long getMicrosecondPosition() {
453         return (long) (getFramePosition() * (1000000.0 / (double) getFormat()
454                 .getSampleRate()));
455     }
456
457     public boolean isActive() {
458         synchronized (control_mutex) {
459             return active;
460         }
461     }
462
463     public boolean isRunning() {
464         synchronized (control_mutex) {
465             return active;
466         }
467     }
468
469     public void start() {
470
471         LineEvent event = null;
472
473         synchronized (control_mutex) {
474             if (isOpen()) {
475                 if (active)
476                     return;
477                 active = true;
478                 active_sg = true;
479                 loopcount = 0;
480                 event = new LineEvent(this, LineEvent.Type.START,
481                         getLongFramePosition());
482             }
483         }
484
485         if (event != null)
486             sendEvent(event);
487     }
488
489     public void stop() {
490         LineEvent event = null;
491
492         synchronized (control_mutex) {
493             if (isOpen()) {
494                 if (!active)
```

```
                    return;
                active = false;
                active_sg = true;
                event = new LineEvent(this, LineEvent.Type.STOP,
                        getLongFramePosition());
            }
        }

        if (event != null)
            sendEvent(event);
    }

    public void close() {
        LineEvent event = null;

        synchronized (control_mutex) {
            if (!isOpen())
                return;
            stop();

            event = new LineEvent(this, LineEvent.Type.CLOSE,
                    getLongFramePosition());

            open = false;
            mixer.getMainMixer().closeLine(this);
        }

        if (event != null)
            sendEvent(event);

    }

    public boolean isOpen() {
        return open;
    }

    public void open() throws LineUnavailableException {
        if (data == null) {
            throw new IllegalArgumentException(
                    "Illegal call to open() in interface Clip");
        }
        open(format, data, offset, bufferSize);
    }

}
```

98 com/sun/media/sound/SoftMixingDataLine.java

```
/*
 * Copyright 2008 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.BooleanControl;
import javax.sound.sampled.Control;
import javax.sound.sampled.DataLine;
import javax.sound.sampled.FloatControl;
import javax.sound.sampled.LineEvent;
import javax.sound.sampled.LineListener;
import javax.sound.sampled.Control.Type;

/**
 * General software mixing line.
 *
 * @author Karl Helgason
 */
public abstract class SoftMixingDataLine implements DataLine {

    public static final FloatControl.Type CHORUS_SEND = new FloatControl.Type(
            "Chorus Send") {
    };

    protected static class AudioFloatInputStreamResampler extends
            AudioFloatInputStream {

        private AudioFloatInputStream ais;

        private AudioFormat targetFormat;

        private float[] skipbuffer;
```

```
61
62          private SoftAbstractResampler resampler;
63
64          private float[] pitch = new float[1];
65
66          private float[] ibuffer2;
67
68          private float[][] ibuffer;
69
70          private float ibuffer_index = 0;
71
72          private int ibuffer_len = 0;
73
74          private int nrofchannels = 0;
75
76          private float[][] cbuffer;
77
78          private int buffer_len = 512;
79
80          private int pad;
81
82          private int pad2;
83
84          private float[] ix = new float[1];
85
86          private int[] ox = new int[1];
87
88          private float[][] mark_ibuffer = null;
89
90          private float mark_ibuffer_index = 0;
91
92          private int mark_ibuffer_len = 0;
93
94          public AudioFloatInputStreamResampler(AudioFloatInputStream ais,
95                  AudioFormat format) {
96              this.ais = ais;
97              AudioFormat sourceFormat = ais.getFormat();
98              targetFormat = new AudioFormat(sourceFormat.getEncoding(), format
99                      .getSampleRate(), sourceFormat.getSampleSizeInBits(),
100                     sourceFormat.getChannels(), sourceFormat.getFrameSize(),
101                     format.getSampleRate(), sourceFormat.isBigEndian());
102             nrofchannels = targetFormat.getChannels();
103             Object interpolation = format.getProperty("interpolation");
104             if (interpolation != null && (interpolation instanceof String)) {
105                 String resamplerType = (String) interpolation;
106                 if (resamplerType.equalsIgnoreCase("point"))
107                     this.resampler = new SoftPointResampler();
108                 if (resamplerType.equalsIgnoreCase("linear"))
109                     this.resampler = new SoftLinearResampler2();
110                 if (resamplerType.equalsIgnoreCase("linear1"))
111                     this.resampler = new SoftLinearResampler();
112                 if (resamplerType.equalsIgnoreCase("linear2"))
113                     this.resampler = new SoftLinearResampler2();
114                 if (resamplerType.equalsIgnoreCase("cubic"))
115                     this.resampler = new SoftCubicResampler();
116                 if (resamplerType.equalsIgnoreCase("lanczos"))
117                     this.resampler = new SoftLanczosResampler();
118                 if (resamplerType.equalsIgnoreCase("sinc"))
119                     this.resampler = new SoftSincResampler();
120             }
121             if (resampler == null)
122                 resampler = new SoftLinearResampler2(); // new
```

```
123             // SoftLinearResampler2();
124             pitch[0] = sourceFormat.getSampleRate() / format.getSampleRate();
125             pad = resampler.getPadding();
126             pad2 = pad * 2;
127             ibuffer = new float[nrofchannels][buffer_len + pad2];
128             ibuffer2 = new float[nrofchannels * buffer_len];
129             ibuffer_index = buffer_len + pad;
130             ibuffer_len = buffer_len;
131         }
132 
133         public int available() throws IOException {
134             return 0;
135         }
136 
137         public void close() throws IOException {
138             ais.close();
139         }
140 
141         public AudioFormat getFormat() {
142             return targetFormat;
143         }
144 
145         public long getFrameLength() {
146             return AudioSystem.NOT_SPECIFIED; // ais.getFrameLength();
147         }
148 
149         public void mark(int readlimit) {
150             ais.mark((int) (readlimit * pitch[0]));
151             mark_ibuffer_index = ibuffer_index;
152             mark_ibuffer_len = ibuffer_len;
153             if (mark_ibuffer == null) {
154                 mark_ibuffer = new float[ibuffer.length][ibuffer[0].length];
155             }
156             for (int c = 0; c < ibuffer.length; c++) {
157                 float[] from = ibuffer[c];
158                 float[] to = mark_ibuffer[c];
159                 for (int i = 0; i < to.length; i++) {
160                     to[i] = from[i];
161                 }
162             }
163         }
164 
165         public boolean markSupported() {
166             return ais.markSupported();
167         }
168 
169         private void readNextBuffer() throws IOException {
170 
171             if (ibuffer_len == -1)
172                 return;
173 
174             for (int c = 0; c < nrofchannels; c++) {
175                 float[] buff = ibuffer[c];
176                 int buffer_len_pad = ibuffer_len + pad2;
177                 for (int i = ibuffer_len, ix = 0; i < buffer_len_pad; i++, ix++) {
178                     buff[ix] = buff[i];
179                 }
180             }
181 
182             ibuffer_index -= (ibuffer_len);
183 
184             ibuffer_len = ais.read(ibuffer2);
```

```
            if (ibuffer_len >= 0) {
                while (ibuffer_len < ibuffer2.length) {
                    int ret = ais.read(ibuffer2, ibuffer_len, ibuffer2.length
                            - ibuffer_len);
                    if (ret == -1)
                        break;
                    ibuffer_len += ret;
                }
                Arrays.fill(ibuffer2, ibuffer_len, ibuffer2.length, 0);
                ibuffer_len /= nrofchannels;
            } else {
                Arrays.fill(ibuffer2, 0, ibuffer2.length, 0);
            }

            int ibuffer2_len = ibuffer2.length;
            for (int c = 0; c < nrofchannels; c++) {
                float[] buff = ibuffer[c];
                for (int i = c, ix = pad2; i < ibuffer2_len; i += nrofchannels, ix++) {
                    buff[ix] = ibuffer2[i];
                }
            }

        }

        public int read(float[] b, int off, int len) throws IOException {

            if (cbuffer == null || cbuffer[0].length < len / nrofchannels) {
                cbuffer = new float[nrofchannels][len / nrofchannels];
            }
            if (ibuffer_len == -1)
                return -1;
            if (len < 0)
                return 0;
            int remain = len / nrofchannels;
            int destPos = 0;
            int in_end = ibuffer_len;
            while (remain > 0) {
                if (ibuffer_len >= 0) {
                    if (ibuffer_index >= (ibuffer_len + pad))
                        readNextBuffer();
                    in_end = ibuffer_len + pad;
                }

                if (ibuffer_len < 0) {
                    in_end = pad2;
                    if (ibuffer_index >= in_end)
                        break;
                }

                if (ibuffer_index < 0)
                    break;
                int preDestPos = destPos;
                for (int c = 0; c < nrofchannels; c++) {
                    ix[0] = ibuffer_index;
                    ox[0] = destPos;
                    float[] buff = ibuffer[c];
                    resampler.interpolate(buff, ix, in_end, pitch, 0,
                            cbuffer[c], ox, len / nrofchannels);
                }
                ibuffer_index = ix[0];
                destPos = ox[0];
                remain -= destPos - preDestPos;
```

```
247             }
248             for (int c = 0; c < nrofchannels; c++) {
249                 int ix = 0;
250                 float[] buff = cbuffer[c];
251                 for (int i = c; i < b.length; i += nrofchannels) {
252                     b[i] = buff[ix++];
253                 }
254             }
255             return len - remain * nrofchannels;
256         }
257 
258         public void reset() throws IOException {
259             ais.reset();
260             if (mark_ibuffer == null)
261                 return;
262             ibuffer_index = mark_ibuffer_index;
263             ibuffer_len = mark_ibuffer_len;
264             for (int c = 0; c < ibuffer.length; c++) {
265                 float[] from = mark_ibuffer[c];
266                 float[] to = ibuffer[c];
267                 for (int i = 0; i < to.length; i++) {
268                     to[i] = from[i];
269                 }
270             }
271 
272         }
273 
274         public long skip(long len) throws IOException {
275             if (len > 0)
276                 return 0;
277             if (skipbuffer == null)
278                 skipbuffer = new float[1024 * targetFormat.getFrameSize()];
279             float[] l_skipbuffer = skipbuffer;
280             long remain = len;
281             while (remain > 0) {
282                 int ret = read(l_skipbuffer, 0, (int) Math.min(remain,
283                         skipbuffer.length));
284                 if (ret < 0) {
285                     if (remain == len)
286                         return ret;
287                     break;
288                 }
289                 remain -= ret;
290             }
291             return len - remain;
292 
293         }
294 
295     }
296 
297     private class Gain extends FloatControl {
298 
299         private Gain() {
300 
301             super(FloatControl.Type.MASTER_GAIN, -80f, 6.0206f, 80f / 128.0f,
302                     -1, 0.0f, "dB", "Minimum", "", "Maximum");
303         }
304 
305         public void setValue(float newValue) {
306             super.setValue(newValue);
307             calcVolume();
308         }
```

```
    }

    private class Mute extends BooleanControl {

        private Mute() {
            super(BooleanControl.Type.MUTE, false, "True", "False");
        }

        public void setValue(boolean newValue) {
            super.setValue(newValue);
            calcVolume();
        }
    }

    private class ApplyReverb extends BooleanControl {

        private ApplyReverb() {
            super(BooleanControl.Type.APPLY_REVERB, false, "True", "False");
        }

        public void setValue(boolean newValue) {
            super.setValue(newValue);
            calcVolume();
        }

    }

    private class Balance extends FloatControl {

        private Balance() {
            super(FloatControl.Type.BALANCE, -1.0f, 1.0f, (1.0f / 128.0f), -1,
                    0.0f, "", "Left", "Center", "Right");
        }

        public void setValue(float newValue) {
            super.setValue(newValue);
            calcVolume();
        }

    }

    private class Pan extends FloatControl {

        private Pan() {
            super(FloatControl.Type.PAN, -1.0f, 1.0f, (1.0f / 128.0f), -1,
                    0.0f, "", "Left", "Center", "Right");
        }

        public void setValue(float newValue) {
            super.setValue(newValue);
            balance_control.setValue(newValue);
        }

        public float getValue() {
            return balance_control.getValue();
        }

    }

    private class ReverbSend extends FloatControl {

        private ReverbSend() {
```

```
            super(FloatControl.Type.REVERB_SEND, -80f, 6.0206f, 80f / 128.0f,
                    -1, -80f, "dB", "Minimum", "", "Maximum");
        }

        public void setValue(float newValue) {
            super.setValue(newValue);
            balance_control.setValue(newValue);
        }

    }

    private class ChorusSend extends FloatControl {

        private ChorusSend() {
            super(CHORUS_SEND, -80f, 6.0206f, 80f / 128.0f, -1, -80f, "dB",
                    "Minimum", "", "Maximum");
        }

        public void setValue(float newValue) {
            super.setValue(newValue);
            balance_control.setValue(newValue);
        }

    }

    private Gain gain_control = new Gain();

    private Mute mute_control = new Mute();

    private Balance balance_control = new Balance();

    private Pan pan_control = new Pan();

    private ReverbSend reverbsend_control = new ReverbSend();

    private ChorusSend chorussend_control = new ChorusSend();

    private ApplyReverb apply_reverb = new ApplyReverb();

    private Control[] controls;

    protected float leftgain = 1;

    protected float rightgain = 1;

    protected float eff1gain = 0;

    protected float eff2gain = 0;

    protected List<LineListener> listeners = new ArrayList<LineListener>();

    protected Object control_mutex;

    protected SoftMixingMixer mixer;

    protected DataLine.Info info;

    protected abstract void processControlLogic();

    protected abstract void processAudioLogic(SoftAudioBuffer[] buffers);

    protected SoftMixingDataLine(SoftMixingMixer mixer, DataLine.Info info) {
```

```
        this.mixer = mixer;
        this.info = info;
        this.control_mutex = mixer.control_mutex;

        controls = new Control[] { gain_control, mute_control, balance_control,
                pan_control, reverbsend_control, chorussend_control,
                apply_reverb };
        calcVolume();
    }

    protected void calcVolume() {
        synchronized (control_mutex) {
            double gain = Math.pow(10.0, gain_control.getValue() / 20.0);
            if (mute_control.getValue())
                gain = 0;
            leftgain = (float) gain;
            rightgain = (float) gain;
            if (mixer.getFormat().getChannels() > 1) {
                // -1 = Left, 0 Center, 1 = Right
                double balance = balance_control.getValue();
                if (balance > 0)
                    leftgain *= (1 - balance);
                else
                    rightgain *= (1 + balance);

            }
        }

        eff1gain = (float) Math.pow(10.0, reverbsend_control.getValue() / 20.0);
        eff2gain = (float) Math.pow(10.0, chorussend_control.getValue() / 20.0);

        if (!apply_reverb.getValue()) {
            eff1gain = 0;
        }
    }

    protected void sendEvent(LineEvent event) {
        if (listeners.size() == 0)
            return;
        LineListener[] listener_array = listeners
                .toArray(new LineListener[listeners.size()]);
        for (LineListener listener : listener_array) {
            listener.update(event);
        }
    }

    public void addLineListener(LineListener listener) {
        synchronized (control_mutex) {
            listeners.add(listener);
        }
    }

    public void removeLineListener(LineListener listener) {
        synchronized (control_mutex) {
            listeners.add(listener);
        }
    }

    public javax.sound.sampled.Line.Info getLineInfo() {
        return info;
    }
```

```
    public Control getControl(Type control) {
        if (control != null) {
            for (int i = 0; i < controls.length; i++) {
                if (controls[i].getType() == control) {
                    return controls[i];
                }
            }
        }
        throw new IllegalArgumentException("Unsupported control type : "
                + control);
    }

    public Control[] getControls() {
        return controls;
    }

    public boolean isControlSupported(Type control) {
        if (control != null) {
            for (int i = 0; i < controls.length; i++) {
                if (controls[i].getType() == control) {
                    return true;
                }
            }
        }
        return false;
    }

}
```

99 com/sun/media/sound/SoftMixingMainMixer.java

```java
/*
 * Copyright 2008 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;

import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;

/**
 * Main mixer for SoftMixingMixer.
 *
 * @author Karl Helgason
 */
public class SoftMixingMainMixer {

    public final static int CHANNEL_LEFT = 0;

    public final static int CHANNEL_RIGHT = 1;

    public final static int CHANNEL_EFFECT1 = 2;

    public final static int CHANNEL_EFFECT2 = 3;

    public final static int CHANNEL_EFFECT3 = 4;

    public final static int CHANNEL_EFFECT4 = 5;

    public final static int CHANNEL_LEFT_DRY = 10;

    public final static int CHANNEL_RIGHT_DRY = 11;

    public final static int CHANNEL_SCRATCH1 = 12;

    public final static int CHANNEL_SCRATCH2 = 13;
```

```
61
62      public final static int CHANNEL_CHANNELMIXER_LEFT = 14;
63
64      public final static int CHANNEL_CHANNELMIXER_RIGHT = 15;
65
66      private SoftMixingMixer mixer;
67
68      private AudioInputStream ais;
69
70      private SoftAudioBuffer[] buffers;
71
72      private SoftAudioProcessor reverb;
73
74      private SoftAudioProcessor chorus;
75
76      private SoftAudioProcessor agc;
77
78      private int nrofchannels;
79
80      private Object control_mutex;
81
82      private List<SoftMixingDataLine> openLinesList = new ArrayList<SoftMixingDataLine>();
83
84      private SoftMixingDataLine[] openLines = new SoftMixingDataLine[0];
85
86      public AudioInputStream getInputStream() {
87          return ais;
88      }
89
90      protected void processAudioBuffers() {
91          for (int i = 0; i < buffers.length; i++) {
92              buffers[i].clear();
93          }
94
95          SoftMixingDataLine[] openLines;
96          synchronized (control_mutex) {
97              openLines = this.openLines;
98              for (int i = 0; i < openLines.length; i++) {
99                  openLines[i].processControlLogic();
100             }
101             chorus.processControlLogic();
102             reverb.processControlLogic();
103             agc.processControlLogic();
104         }
105         for (int i = 0; i < openLines.length; i++) {
106             openLines[i].processAudioLogic(buffers);
107         }
108
109         chorus.processAudio();
110         reverb.processAudio();
111
112         agc.processAudio();
113
114     }
115
116     public SoftMixingMainMixer(SoftMixingMixer mixer) {
117         this.mixer = mixer;
118
119         nrofchannels = mixer.getFormat().getChannels();
120
121         int buffersize = (int) (mixer.getFormat().getSampleRate() / mixer
122                 .getControlRate());
```

```
123
124         control_mutex = mixer.control_mutex;
125         buffers = new SoftAudioBuffer[16];
126         for (int i = 0; i < buffers.length; i++) {
127             buffers[i] = new SoftAudioBuffer(buffersize, mixer.getFormat());
128
129         }
130
131         reverb = new SoftReverb();
132         chorus = new SoftChorus();
133         agc = new SoftLimiter();
134
135         float samplerate = mixer.getFormat().getSampleRate();
136         float controlrate = mixer.getControlRate();
137         reverb.init(samplerate, controlrate);
138         chorus.init(samplerate, controlrate);
139         agc.init(samplerate, controlrate);
140
141         reverb.setMixMode(true);
142         chorus.setMixMode(true);
143         agc.setMixMode(false);
144
145         chorus.setInput(0, buffers[CHANNEL_EFFECT2]);
146         chorus.setOutput(0, buffers[CHANNEL_LEFT]);
147         if (nrofchannels != 1)
148             chorus.setOutput(1, buffers[CHANNEL_RIGHT]);
149         chorus.setOutput(2, buffers[CHANNEL_EFFECT1]);
150
151         reverb.setInput(0, buffers[CHANNEL_EFFECT1]);
152         reverb.setOutput(0, buffers[CHANNEL_LEFT]);
153         if (nrofchannels != 1)
154             reverb.setOutput(1, buffers[CHANNEL_RIGHT]);
155
156         agc.setInput(0, buffers[CHANNEL_LEFT]);
157         if (nrofchannels != 1)
158             agc.setInput(1, buffers[CHANNEL_RIGHT]);
159         agc.setOutput(0, buffers[CHANNEL_LEFT]);
160         if (nrofchannels != 1)
161             agc.setOutput(1, buffers[CHANNEL_RIGHT]);
162
163         InputStream in = new InputStream() {
164
165             private SoftAudioBuffer[] buffers = SoftMixingMainMixer.this.buffers;
166
167             private int nrofchannels = SoftMixingMainMixer.this.mixer
168                     .getFormat().getChannels();
169
170             private int buffersize = buffers[0].getSize();
171
172             private byte[] bbuffer = new byte[buffersize
173                     * (SoftMixingMainMixer.this.mixer.getFormat()
174                             .getSampleSizeInBits() / 8) * nrofchannels];
175
176             private int bbuffer_pos = 0;
177
178             private byte[] single = new byte[1];
179
180             public void fillBuffer() {
181                 processAudioBuffers();
182                 for (int i = 0; i < nrofchannels; i++)
183                     buffers[i].get(bbuffer, i);
184                 bbuffer_pos = 0;
```

```
            }

            public int read(byte[] b, int off, int len) {
                int bbuffer_len = bbuffer.length;
                int offlen = off + len;
                byte[] bbuffer = this.bbuffer;
                while (off < offlen)
                    if (available() == 0)
                        fillBuffer();
                    else {
                        int bbuffer_pos = this.bbuffer_pos;
                        while (off < offlen && bbuffer_pos < bbuffer_len)
                            b[off++] = bbuffer[bbuffer_pos++];
                        this.bbuffer_pos = bbuffer_pos;
                    }
                return len;
            }

            public int read() throws IOException {
                int ret = read(single);
                if (ret == -1)
                    return -1;
                return single[0] & 0xFF;
            }

            public int available() {
                return bbuffer.length - bbuffer_pos;
            }

            public void close() {
                SoftMixingMainMixer.this.mixer.close();
            }

        };

        ais = new AudioInputStream(in, mixer.getFormat(),
                AudioSystem.NOT_SPECIFIED);

    }

    public void openLine(SoftMixingDataLine line) {
        synchronized (control_mutex) {
            openLinesList.add(line);
            openLines = openLinesList
                    .toArray(new SoftMixingDataLine[openLinesList.size()]);
        }
    }

    public void closeLine(SoftMixingDataLine line) {
        synchronized (control_mutex) {
            openLinesList.remove(line);
            openLines = openLinesList
                    .toArray(new SoftMixingDataLine[openLinesList.size()]);
            if (openLines.length == 0)
                if (mixer.implicitOpen)
                    mixer.close();
        }

    }

    public SoftMixingDataLine[] getOpenLines() {
        synchronized (control_mutex) {
```

```
247                 return openLines;
248             }
249 
250         }
251 
252         public void close() {
253             SoftMixingDataLine[] openLines = this.openLines;
254             for (int i = 0; i < openLines.length; i++) {
255                 openLines[i].close();
256             }
257         }
258 
259 }
```

100 com/sun/media/sound/SoftMixingMixer.java

```
/*
 * Copyright 2008 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.Clip;
import javax.sound.sampled.Control;
import javax.sound.sampled.DataLine;
import javax.sound.sampled.Line;
import javax.sound.sampled.LineEvent;
import javax.sound.sampled.LineListener;
import javax.sound.sampled.LineUnavailableException;
import javax.sound.sampled.Mixer;
import javax.sound.sampled.SourceDataLine;
import javax.sound.sampled.AudioFormat.Encoding;
import javax.sound.sampled.Control.Type;

/**
 * Software audio mixer
 *
 * @author Karl Helgason
 */
public class SoftMixingMixer implements Mixer {

    private static class Info extends Mixer.Info {
        public Info() {
            super(INFO_NAME, INFO_VENDOR, INFO_DESCRIPTION, INFO_VERSION);
        }
    }

    protected static final String INFO_NAME = "Gervill Sound Mixer";

```

```
61      protected static final String INFO_VENDOR = "OpenJDK Proposal";
62
63      protected static final String INFO_DESCRIPTION = "Software Sound Mixer";
64
65      protected static final String INFO_VERSION = "1.0";
66
67      protected final static Mixer.Info info = new Info();
68
69      protected Object control_mutex = this;
70
71      protected boolean implicitOpen = false;
72
73      private boolean open = false;
74
75      private SoftMixingMainMixer mainmixer = null;
76
77      private AudioFormat format = new AudioFormat(44100, 16, 2, true, false);
78
79      private SourceDataLine sourceDataLine = null;
80
81      private SoftAudioPusher pusher = null;
82
83      private AudioInputStream pusher_stream = null;
84
85      private float controlrate = 147f;
86
87      private long latency = 100000; // 100 msec
88
89      private boolean jitter_correction = false;
90
91      private List<LineListener> listeners = new ArrayList<LineListener>();
92
93      private javax.sound.sampled.Line.Info[] sourceLineInfo;
94
95      public SoftMixingMixer() {
96
97          sourceLineInfo = new javax.sound.sampled.Line.Info[2];
98
99          ArrayList<AudioFormat> formats = new ArrayList<AudioFormat>();
100         for (int channels = 1; channels <= 2; channels++) {
101             formats.add(new AudioFormat(Encoding.PCM_SIGNED,
102                     AudioSystem.NOT_SPECIFIED, 8, channels, channels,
103                     AudioSystem.NOT_SPECIFIED, false));
104             formats.add(new AudioFormat(Encoding.PCM_UNSIGNED,
105                     AudioSystem.NOT_SPECIFIED, 8, channels, channels,
106                     AudioSystem.NOT_SPECIFIED, false));
107             for (int bits = 16; bits < 32; bits += 8) {
108                 formats.add(new AudioFormat(Encoding.PCM_SIGNED,
109                         AudioSystem.NOT_SPECIFIED, bits, channels, channels
110                                 * bits / 8, AudioSystem.NOT_SPECIFIED, false));
111                 formats.add(new AudioFormat(Encoding.PCM_UNSIGNED,
112                         AudioSystem.NOT_SPECIFIED, bits, channels, channels
113                                 * bits / 8, AudioSystem.NOT_SPECIFIED, false));
114                 formats.add(new AudioFormat(Encoding.PCM_SIGNED,
115                         AudioSystem.NOT_SPECIFIED, bits, channels, channels
116                                 * bits / 8, AudioSystem.NOT_SPECIFIED, true));
117                 formats.add(new AudioFormat(Encoding.PCM_UNSIGNED,
118                         AudioSystem.NOT_SPECIFIED, bits, channels, channels
119                                 * bits / 8, AudioSystem.NOT_SPECIFIED, true));
120             }
121             formats.add(new AudioFormat(AudioFloatConverter.PCM_FLOAT,
122                     AudioSystem.NOT_SPECIFIED, 32, channels, channels * 4,
```

```
                AudioSystem.NOT_SPECIFIED, false));
        formats.add(new AudioFormat(AudioFloatConverter.PCM_FLOAT,
                AudioSystem.NOT_SPECIFIED, 32, channels, channels * 4,
                AudioSystem.NOT_SPECIFIED, true));
        formats.add(new AudioFormat(AudioFloatConverter.PCM_FLOAT,
                AudioSystem.NOT_SPECIFIED, 64, channels, channels * 8,
                AudioSystem.NOT_SPECIFIED, false));
        formats.add(new AudioFormat(AudioFloatConverter.PCM_FLOAT,
                AudioSystem.NOT_SPECIFIED, 64, channels, channels * 8,
                AudioSystem.NOT_SPECIFIED, true));
    }
    AudioFormat[] formats_array = formats.toArray(new AudioFormat[formats
            .size()]);
    sourceLineInfo[0] = new DataLine.Info(SourceDataLine.class,
            formats_array, AudioSystem.NOT_SPECIFIED,
            AudioSystem.NOT_SPECIFIED);
    sourceLineInfo[1] = new DataLine.Info(Clip.class, formats_array,
            AudioSystem.NOT_SPECIFIED, AudioSystem.NOT_SPECIFIED);
}

public Line getLine(Line.Info info) throws LineUnavailableException {

    if (!isLineSupported(info))
        throw new IllegalArgumentException("Line unsupported: " + info);

    if ((info.getLineClass() == SourceDataLine.class)) {
        return new SoftMixingSourceDataLine(this, (DataLine.Info) info);
    }
    if ((info.getLineClass() == Clip.class)) {
        return new SoftMixingClip(this, (DataLine.Info) info);
    }

    throw new IllegalArgumentException("Line unsupported: " + info);
}

public int getMaxLines(Line.Info info) {
    if (info.getLineClass() == SourceDataLine.class)
        return AudioSystem.NOT_SPECIFIED;
    if (info.getLineClass() == Clip.class)
        return AudioSystem.NOT_SPECIFIED;
    return 0;
}

public javax.sound.sampled.Mixer.Info getMixerInfo() {
    return info;
}

public javax.sound.sampled.Line.Info[] getSourceLineInfo() {
    Line.Info[] localArray = new Line.Info[sourceLineInfo.length];
    System.arraycopy(sourceLineInfo, 0, localArray, 0,
            sourceLineInfo.length);
    return localArray;
}

public javax.sound.sampled.Line.Info[] getSourceLineInfo(
        javax.sound.sampled.Line.Info info) {
    int i;
    ArrayList<javax.sound.sampled.Line.Info> infos = new ArrayList<javax.sound.sampled.Line.
       Info>();

    for (i = 0; i < sourceLineInfo.length; i++) {
        if (info.matches(sourceLineInfo[i])) {
```

```
                infos.add(sourceLineInfo[i]);
            }
        }
        return infos.toArray(new Line.Info[infos.size()]);
    }

    public Line[] getSourceLines() {

        Line[] localLines;

        synchronized (control_mutex) {

            if (mainmixer == null)
                return new Line[0];
            SoftMixingDataLine[] sourceLines = mainmixer.getOpenLines();

            localLines = new Line[sourceLines.length];

            for (int i = 0; i < localLines.length; i++) {
                localLines[i] = sourceLines[i];
            }
        }

        return localLines;
    }

    public javax.sound.sampled.Line.Info[] getTargetLineInfo() {
        return new javax.sound.sampled.Line.Info[0];
    }

    public javax.sound.sampled.Line.Info[] getTargetLineInfo(
            javax.sound.sampled.Line.Info info) {
        return new javax.sound.sampled.Line.Info[0];
    }

    public Line[] getTargetLines() {
        return new Line[0];
    }

    public boolean isLineSupported(javax.sound.sampled.Line.Info info) {
        if (info != null) {
            for (int i = 0; i < sourceLineInfo.length; i++) {
                if (info.matches(sourceLineInfo[i])) {
                    return true;
                }
            }
        }
        return false;
    }

    public boolean isSynchronizationSupported(Line[] lines, boolean maintainSync) {
        return false;
    }

    public void synchronize(Line[] lines, boolean maintainSync) {
        throw new IllegalArgumentException(
                "Synchronization not supported by this mixer.");
    }

    public void unsynchronize(Line[] lines) {
        throw new IllegalArgumentException(
                "Synchronization not supported by this mixer.");
```

```
246     }
247
248     public void addLineListener(LineListener listener) {
249         synchronized (control_mutex) {
250             listeners.add(listener);
251         }
252     }
253
254     private void sendEvent(LineEvent event) {
255         if (listeners.size() == 0)
256             return;
257         LineListener[] listener_array = listeners
258                 .toArray(new LineListener[listeners.size()]);
259         for (LineListener listener : listener_array) {
260             listener.update(event);
261         }
262     }
263
264     public void close() {
265         if (!isOpen())
266             return;
267
268         sendEvent(new LineEvent(this, LineEvent.Type.CLOSE,
269                 AudioSystem.NOT_SPECIFIED));
270
271         SoftAudioPusher pusher_to_be_closed = null;
272         AudioInputStream pusher_stream_to_be_closed = null;
273         synchronized (control_mutex) {
274             if (pusher != null) {
275                 pusher_to_be_closed = pusher;
276                 pusher_stream_to_be_closed = pusher_stream;
277                 pusher = null;
278                 pusher_stream = null;
279             }
280         }
281
282         if (pusher_to_be_closed != null) {
283             // Pusher must not be closed synchronized against control_mutex
284             // this may result in synchronized conflict between pusher and
285             // current thread.
286             pusher_to_be_closed.stop();
287
288             try {
289                 pusher_stream_to_be_closed.close();
290             } catch (IOException e) {
291                 e.printStackTrace();
292             }
293         }
294
295         synchronized (control_mutex) {
296
297             if (mainmixer != null)
298                 mainmixer.close();
299             open = false;
300
301             if (sourceDataLine != null) {
302                 sourceDataLine.drain();
303                 sourceDataLine.close();
304                 sourceDataLine = null;
305             }
306
307         }
```

```
308
309     }
310
311     public Control getControl(Type control) {
312         throw new IllegalArgumentException("Unsupported control type : "
313                 + control);
314     }
315
316     public Control[] getControls() {
317         return new Control[0];
318     }
319
320     public javax.sound.sampled.Line.Info getLineInfo() {
321         return new Line.Info(Mixer.class);
322     }
323
324     public boolean isControlSupported(Type control) {
325         return false;
326     }
327
328     public boolean isOpen() {
329         synchronized (control_mutex) {
330             return open;
331         }
332     }
333
334     public void open() throws LineUnavailableException {
335         if (isOpen()) {
336             implicitOpen = false;
337             return;
338         }
339         open(null);
340     }
341
342     public void open(SourceDataLine line) throws LineUnavailableException {
343         if (isOpen()) {
344             implicitOpen = false;
345             return;
346         }
347         synchronized (control_mutex) {
348
349             try {
350
351                 if (line != null)
352                     format = line.getFormat();
353
354                 AudioInputStream ais = openStream(getFormat());
355
356                 if (line == null) {
357                     synchronized (SoftMixingMixerProvider.mutex) {
358                         SoftMixingMixerProvider.lockthread = Thread
359                                 .currentThread();
360                     }
361
362                     try {
363                         Mixer defaultmixer = AudioSystem.getMixer(null);
364                         if (defaultmixer != null)
365                         {
366                             // Search for suitable line
367
368                             DataLine.Info idealinfo = null;
369                             AudioFormat idealformat = null;
```

```
370
371                     Line.Info[] lineinfos = defaultmixer.getSourceLineInfo();
372                     idealFound:
373                     for (int i = 0; i < lineinfos.length; i++) {
374                         if(lineinfos[i].getLineClass() == SourceDataLine.class)
375                         {
376                             DataLine.Info info = (DataLine.Info)lineinfos[i];
377                             AudioFormat[] formats = info.getFormats();
378                             for (int j = 0; j < formats.length; j++) {
379                                 AudioFormat format = formats[j];
380                                 if(format.getChannels() == 2 ||
381                                         format.getChannels() == AudioSystem.NOT_SPECIFIED
                                            )
382                                 if(format.getEncoding().equals(Encoding.PCM_SIGNED) ||
383                                         format.getEncoding().equals(Encoding.PCM_UNSIGNED
                                            ))
384                                 if(format.getSampleRate() == AudioSystem.NOT_SPECIFIED ||
385                                         format.getSampleRate() == 48000.0)
386                                 if(format.getSampleSizeInBits() == AudioSystem.
                                    NOT_SPECIFIED ||
387                                         format.getSampleSizeInBits() == 16)
388                                 {
389                                     idealinfo = info;
390                                     int ideal_channels = format.getChannels();
391                                     boolean ideal_signed = format.getEncoding().equals(
                                            Encoding.PCM_SIGNED);
392                                     float ideal_rate = format.getSampleRate();
393                                     boolean ideal_endian = format.isBigEndian();
394                                     int ideal_bits = format.getSampleSizeInBits();
395                                     if(ideal_bits == AudioSystem.NOT_SPECIFIED)
                                            ideal_bits = 16;
396                                     if(ideal_channels == AudioSystem.NOT_SPECIFIED)
                                            ideal_channels = 2;
397                                     if(ideal_rate == AudioSystem.NOT_SPECIFIED)
                                            ideal_rate = 48000;
398                                     idealformat = new AudioFormat(ideal_rate, ideal_bits,
399                                             ideal_channels, ideal_signed, ideal_endian);
400                                     break idealFound;
401                                 }
402                             }
403                         }
404                     }
405
406                     if(idealformat != null)
407                     {
408                         format = idealformat;
409                         line = (SourceDataLine) defaultmixer.getLine(idealinfo);
410                     }
411                 }
412
413                 if(line == null)
414                     line = AudioSystem.getSourceDataLine(format);
415             } finally {
416                 synchronized (SoftMixingMixerProvider.mutex) {
417                     SoftMixingMixerProvider.lockthread = null;
418                 }
419             }
420
421             if (line == null)
422                 throw new IllegalArgumentException("No␣line␣matching␣"
423                         + info.toString() + "␣is␣supported.");
424         }
```

```
425
426                 double latency = this.latency;
427
428                 if (!line.isOpen()) {
429                     int bufferSize = getFormat().getFrameSize()
430                             * (int) (getFormat().getFrameRate() * (latency / 1000000f));
431                     line.open(getFormat(), bufferSize);
432
433                     // Remember that we opened that line
434                     // so we can close again in SoftSynthesizer.close()
435                     sourceDataLine = line;
436                 }
437                 if (!line.isActive())
438                     line.start();
439
440                 int controlbuffersize = 512;
441                 try {
442                     controlbuffersize = ais.available();
443                 } catch (IOException e) {
444                 }
445
446                 // Tell mixer not fill read buffers fully.
447                 // This lowers latency, and tells DataPusher
448                 // to read in smaller amounts.
449                 // mainmixer.readfully = false;
450                 // pusher = new DataPusher(line, ais);
451
452                 int buffersize = line.getBufferSize();
453                 buffersize -= buffersize % controlbuffersize;
454
455                 if (buffersize < 3 * controlbuffersize)
456                     buffersize = 3 * controlbuffersize;
457
458                 if (jitter_correction) {
459                     ais = new SoftJitterCorrector(ais, buffersize,
460                             controlbuffersize);
461                 }
462                 pusher = new SoftAudioPusher(line, ais, controlbuffersize);
463                 pusher_stream = ais;
464                 pusher.start();
465
466             } catch (LineUnavailableException e) {
467                 if (isOpen())
468                     close();
469                 throw new LineUnavailableException(e.toString());
470             }
471
472         }
473     }
474
475     public AudioInputStream openStream(AudioFormat targetFormat)
476             throws LineUnavailableException {
477
478         if (isOpen())
479             throw new LineUnavailableException("Mixer is already open");
480
481         synchronized (control_mutex) {
482
483             open = true;
484
485             implicitOpen = false;
486
```

```
            if (targetFormat != null)
                format = targetFormat;

            mainmixer = new SoftMixingMainMixer(this);

            sendEvent(new LineEvent(this, LineEvent.Type.OPEN,
                    AudioSystem.NOT_SPECIFIED));

            return mainmixer.getInputStream();

        }

    }

    public void removeLineListener(LineListener listener) {
        synchronized (control_mutex) {
            listeners.remove(listener);
        }
    }

    public long getLatency() {
        synchronized (control_mutex) {
            return latency;
        }
    }

    public AudioFormat getFormat() {
        synchronized (control_mutex) {
            return format;
        }
    }

    protected float getControlRate() {
        return controlrate;
    }

    protected SoftMixingMainMixer getMainMixer() {
        if (!isOpen())
            return null;
        return mainmixer;
    }

}
```

101 com/sun/media/sound/SoftMixingMixerProvider.java

```java
/*
 * Copyright 2008 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import javax.sound.sampled.Mixer;
import javax.sound.sampled.Mixer.Info;
import javax.sound.sampled.spi.MixerProvider;

/**
 * Provider for software audio mixer
 *
 * @author Karl Helgason
 */
public class SoftMixingMixerProvider extends MixerProvider {

    static SoftMixingMixer globalmixer = null;

    static Thread lockthread = null;

    protected final static Object mutex = new Object();

    public Mixer getMixer(Info info) {
        if (!(info == null || info == SoftMixingMixer.info)) {
            throw new IllegalArgumentException("Mixer " + info.toString()
                    + " not supported by this provider.");
        }
        synchronized (mutex) {
            if (lockthread != null)
                if (Thread.currentThread() == lockthread)
                    throw new IllegalArgumentException("Mixer "
                            + info.toString()
                            + " not supported by this provider.");
            if (globalmixer == null)
                globalmixer = new SoftMixingMixer();
            return globalmixer;
        }

    }
```

```
61
62      public Info[] getMixerInfo() {
63          return new Info[] { SoftMixingMixer.info };
64      }
65
66  }
```

102 com/sun/media/sound/SoftMixingSourceDataLine.java

```
/*
 * Copyright 2008 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.io.IOException;
import java.io.InputStream;
import java.util.Arrays;

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

/**
 * SourceDataLine implemention for the SoftMixingMixer.
 *
 * @author Karl Helgason
 */
public class SoftMixingSourceDataLine extends SoftMixingDataLine implements
        SourceDataLine {

    private boolean open = false;

    private AudioFormat format = new AudioFormat(44100.0f, 16, 2, true, false);

    private int framesize;

    private int bufferSize = -1;

    private float[] readbuffer;

    private boolean active = false;

    private byte[] cycling_buffer;

```

```
    private int cycling_read_pos = 0;

    private int cycling_write_pos = 0;

    private int cycling_avail = 0;

    private long cycling_framepos = 0;

    private AudioFloatInputStream afis;

    private static class NonBlockingFloatInputStream extends
            AudioFloatInputStream {
        AudioFloatInputStream ais;

        public NonBlockingFloatInputStream(AudioFloatInputStream ais) {
            this.ais = ais;
        }

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

        public void close() throws IOException {
            ais.close();
        }

        public AudioFormat getFormat() {
            return ais.getFormat();
        }

        public long getFrameLength() {
            return ais.getFrameLength();
        }

        public void mark(int readlimit) {
            ais.mark(readlimit);
        }

        public boolean markSupported() {
            return ais.markSupported();
        }

        public int read(float[] b, int off, int len) throws IOException {
            int avail = available();
            if (len > avail) {
                int ret = ais.read(b, off, avail);
                Arrays.fill(b, off + ret, off + len, 0);
                return len;
            }
            return ais.read(b, off, len);
        }

        public void reset() throws IOException {
            ais.reset();
        }

        public long skip(long len) throws IOException {
            return ais.skip(len);
        }

    }

```

```
123     protected SoftMixingSourceDataLine(SoftMixingMixer mixer, DataLine.Info info) {
124         super(mixer, info);
125     }
126 
127     public int write(byte[] b, int off, int len) {
128         if (!isOpen())
129             return 0;
130         if (len % framesize != 0)
131             throw new IllegalArgumentException(
132                     "Number of bytes does not represent an integral number of sample frames.");
133 
134         byte[] buff = cycling_buffer;
135         int buff_len = cycling_buffer.length;
136 
137         int l = 0;
138         while (l != len) {
139             int avail;
140             synchronized (cycling_buffer) {
141                 int pos = cycling_write_pos;
142                 avail = cycling_avail;
143                 while (l != len) {
144                     if (avail == buff_len)
145                         break;
146                     buff[pos++] = b[off++];
147                     l++;
148                     avail++;
149                     if (pos == buff_len)
150                         pos = 0;
151                 }
152                 cycling_avail = avail;
153                 cycling_write_pos = pos;
154                 if (l == len)
155                     return l;
156             }
157             if (avail == buff_len) {
158                 try {
159                     Thread.sleep(1);
160                 } catch (InterruptedException e) {
161                     return l;
162                 }
163                 if (!isRunning())
164                     return l;
165             }
166         }
167 
168         return l;
169     }
170 
171     //
172     // BooleanControl.Type.APPLY_REVERB
173     // BooleanControl.Type.MUTE
174     // EnumControl.Type.REVERB
175     //
176     // FloatControl.Type.SAMPLE_RATE
177     // FloatControl.Type.REVERB_SEND
178     // FloatControl.Type.VOLUME
179     // FloatControl.Type.PAN
180     // FloatControl.Type.MASTER_GAIN
181     // FloatControl.Type.BALANCE
182 
183     private boolean _active = false;
184 
```

```
185     private AudioFormat outputformat;
186
187     private int out_nrofchannels;
188
189     private int in_nrofchannels;
190
191     private float _rightgain;
192
193     private float _leftgain;
194
195     private float _eff1gain;
196
197     private float _eff2gain;
198
199     protected void processControlLogic() {
200         _active = active;
201         _rightgain = rightgain;
202         _leftgain = leftgain;
203         _eff1gain = eff1gain;
204         _eff2gain = eff2gain;
205     }
206
207     protected void processAudioLogic(SoftAudioBuffer[] buffers) {
208         if (_active) {
209             float[] left = buffers[SoftMixingMainMixer.CHANNEL_LEFT].array();
210             float[] right = buffers[SoftMixingMainMixer.CHANNEL_RIGHT].array();
211             int bufferlen = buffers[SoftMixingMainMixer.CHANNEL_LEFT].getSize();
212
213             int readlen = bufferlen * in_nrofchannels;
214             if (readbuffer == null || readbuffer.length < readlen) {
215                 readbuffer = new float[readlen];
216             }
217             int ret = 0;
218             try {
219                 ret = afis.read(readbuffer);
220                 if (ret != in_nrofchannels)
221                     Arrays.fill(readbuffer, ret, readlen, 0);
222             } catch (IOException e) {
223             }
224
225             int in_c = in_nrofchannels;
226             for (int i = 0, ix = 0; i < bufferlen; i++, ix += in_c) {
227                 left[i] += readbuffer[ix] * _leftgain;
228             }
229             if (out_nrofchannels != 1) {
230                 if (in_nrofchannels == 1) {
231                     for (int i = 0, ix = 0; i < bufferlen; i++, ix += in_c) {
232                         right[i] += readbuffer[ix] * _rightgain;
233                     }
234                 } else {
235                     for (int i = 0, ix = 1; i < bufferlen; i++, ix += in_c) {
236                         right[i] += readbuffer[ix] * _rightgain;
237                     }
238                 }
239
240             }
241
242             if (_eff1gain > 0.0001) {
243                 float[] eff1 = buffers[SoftMixingMainMixer.CHANNEL_EFFECT1]
244                         .array();
245                 for (int i = 0, ix = 0; i < bufferlen; i++, ix += in_c) {
246                     eff1[i] += readbuffer[ix] * _eff1gain;
```

```
247                 }
248                 if (in_nrofchannels == 2) {
249                     for (int i = 0, ix = 1; i < bufferlen; i++, ix += in_c) {
250                         eff1[i] += readbuffer[ix] * _eff1gain;
251                     }
252                 }
253             }
254 
255             if (_eff2gain > 0.0001) {
256                 float[] eff2 = buffers[SoftMixingMainMixer.CHANNEL_EFFECT2]
257                         .array();
258                 for (int i = 0, ix = 0; i < bufferlen; i++, ix += in_c) {
259                     eff2[i] += readbuffer[ix] * _eff2gain;
260                 }
261                 if (in_nrofchannels == 2) {
262                     for (int i = 0, ix = 1; i < bufferlen; i++, ix += in_c) {
263                         eff2[i] += readbuffer[ix] * _eff2gain;
264                     }
265                 }
266             }
267 
268         }
269     }
270 
271     public void open() throws LineUnavailableException {
272         open(format);
273     }
274 
275     public void open(AudioFormat format) throws LineUnavailableException {
276         if (bufferSize == -1)
277             bufferSize = ((int) (format.getFrameRate() / 2))
278                     * format.getFrameSize();
279         open(format, bufferSize);
280     }
281 
282     public void open(AudioFormat format, int bufferSize)
283             throws LineUnavailableException {
284 
285         LineEvent event = null;
286 
287         if (bufferSize < format.getFrameSize() * 32)
288             bufferSize = format.getFrameSize() * 32;
289 
290         synchronized (control_mutex) {
291 
292             if (!isOpen()) {
293                 if (!mixer.isOpen()) {
294                     mixer.open();
295                     mixer.implicitOpen = true;
296                 }
297 
298                 event = new LineEvent(this, LineEvent.Type.OPEN, 0);
299 
300                 this.bufferSize = bufferSize - bufferSize
301                         % format.getFrameSize();
302                 this.format = format;
303                 this.framesize = format.getFrameSize();
304                 this.outputformat = mixer.getFormat();
305                 out_nrofchannels = outputformat.getChannels();
306                 in_nrofchannels = format.getChannels();
307 
308                 open = true;
```

```
309
310                 mixer.getMainMixer().openLine(this);
311
312                 cycling_buffer = new byte[framesize * bufferSize];
313                 cycling_read_pos = 0;
314                 cycling_write_pos = 0;
315                 cycling_avail = 0;
316                 cycling_framepos = 0;
317
318                 InputStream cycling_inputstream = new InputStream() {
319
320                     public int read() throws IOException {
321                         byte[] b = new byte[1];
322                         int ret = read(b);
323                         if (ret < 0)
324                             return ret;
325                         return b[0] & 0xFF;
326                     }
327
328                     public int available() throws IOException {
329                         synchronized (cycling_buffer) {
330                             return cycling_avail;
331                         }
332                     }
333
334                     public int read(byte[] b, int off, int len)
335                             throws IOException {
336
337                         synchronized (cycling_buffer) {
338                             if (len > cycling_avail)
339                                 len = cycling_avail;
340                             int pos = cycling_read_pos;
341                             byte[] buff = cycling_buffer;
342                             int buff_len = buff.length;
343                             for (int i = 0; i < len; i++) {
344                                 b[off++] = buff[pos];
345                                 pos++;
346                                 if (pos == buff_len)
347                                     pos = 0;
348                             }
349                             cycling_read_pos = pos;
350                             cycling_avail -= len;
351                             cycling_framepos += len / framesize;
352                         }
353                         return len;
354                     }
355
356                 };
357
358                 afis = AudioFloatInputStream
359                         .getInputStream(new AudioInputStream(
360                                 cycling_inputstream, format,
361                                 AudioSystem.NOT_SPECIFIED));
362                 afis = new NonBlockingFloatInputStream(afis);
363
364                 if (Math.abs(format.getSampleRate()
365                         - outputformat.getSampleRate()) > 0.000001)
366                     afis = new AudioFloatInputStreamResampler(afis,
367                             outputformat);
368
369             } else {
370                 if (!format.matches(getFormat())) {
```

```
                    throw new IllegalStateException(
                            "Line is already open with format " + getFormat()
                                    + " and bufferSize " + getBufferSize());
                }
            }

        }

        if (event != null)
            sendEvent(event);

    }

    public int available() {
        synchronized (cycling_buffer) {
            return cycling_buffer.length - cycling_avail;
        }
    }

    public void drain() {
        while (true) {
            int avail;
            synchronized (cycling_buffer) {
                avail = cycling_avail;
            }
            if (avail != 0)
                return;
            try {
                Thread.sleep(1);
            } catch (InterruptedException e) {
                return;
            }
        }
    }

    public void flush() {
        synchronized (cycling_buffer) {
            cycling_read_pos = 0;
            cycling_write_pos = 0;
            cycling_avail = 0;
        }
    }

    public int getBufferSize() {
        synchronized (control_mutex) {
            return bufferSize;
        }
    }

    public AudioFormat getFormat() {
        synchronized (control_mutex) {
            return format;
        }
    }

    public int getFramePosition() {
        return (int) getLongFramePosition();
    }

    public float getLevel() {
        return AudioSystem.NOT_SPECIFIED;
    }
```

```
433
434     public long getLongFramePosition() {
435         synchronized (cycling_buffer) {
436             return cycling_framepos;
437         }
438     }
439
440     public long getMicrosecondPosition() {
441         return (long) (getLongFramePosition() * (1000000.0 / (double) getFormat()
442                 .getSampleRate()));
443     }
444
445     public boolean isActive() {
446         synchronized (control_mutex) {
447             return active;
448         }
449     }
450
451     public boolean isRunning() {
452         synchronized (control_mutex) {
453             return active;
454         }
455     }
456
457     public void start() {
458
459         LineEvent event = null;
460
461         synchronized (control_mutex) {
462             if (isOpen()) {
463                 if (active)
464                     return;
465                 active = true;
466                 event = new LineEvent(this, LineEvent.Type.START,
467                         getLongFramePosition());
468             }
469         }
470
471         if (event != null)
472             sendEvent(event);
473     }
474
475     public void stop() {
476         LineEvent event = null;
477
478         synchronized (control_mutex) {
479             if (isOpen()) {
480                 if (!active)
481                     return;
482                 active = false;
483                 event = new LineEvent(this, LineEvent.Type.STOP,
484                         getLongFramePosition());
485             }
486         }
487
488         if (event != null)
489             sendEvent(event);
490     }
491
492     public void close() {
493
494         LineEvent event = null;
```

```
495
496         synchronized (control_mutex) {
497             if (!isOpen())
498                 return;
499             stop();
500
501             event = new LineEvent(this, LineEvent.Type.CLOSE,
502                     getLongFramePosition());
503
504             open = false;
505             mixer.getMainMixer().closeLine(this);
506         }
507
508         if (event != null)
509             sendEvent(event);
510
511     }
512
513     public boolean isOpen() {
514         synchronized (control_mutex) {
515             return open;
516         }
517     }
518
519 }
```

103 com/sun/media/sound/SoftPerformer.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Comparator;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * This class decodes information from ModelPeformer for use in SoftVoice.
 * It also adds default connections if they where missing in ModelPerformer.
 *
 * @author Karl Helgason
 */
public class SoftPerformer {

    static ModelConnectionBlock[] defaultconnections
            = new ModelConnectionBlock[42];

    static {
        int o = 0;
        defaultconnections[o++] = new ModelConnectionBlock(
            new ModelSource(
                new ModelIdentifier("noteon", "on", 0),
                ModelStandardTransform.DIRECTION_MIN2MAX,
                ModelStandardTransform.POLARITY_UNIPOLAR,
                ModelStandardTransform.TRANSFORM_LINEAR),
            1, new ModelDestination(new ModelIdentifier("eg", "on", 0)));

        defaultconnections[o++] = new ModelConnectionBlock(
            new ModelSource(
                new ModelIdentifier("noteon", "on", 0),
                ModelStandardTransform.DIRECTION_MIN2MAX,
                ModelStandardTransform.POLARITY_UNIPOLAR,
                ModelStandardTransform.TRANSFORM_LINEAR),
```

```
            1, new ModelDestination(new ModelIdentifier("eg", "on", 1)));

        defaultconnections[o++] = new ModelConnectionBlock(
            new ModelSource(
                new ModelIdentifier("eg", "active", 0),
                ModelStandardTransform.DIRECTION_MIN2MAX,
                ModelStandardTransform.POLARITY_UNIPOLAR,
                ModelStandardTransform.TRANSFORM_LINEAR),
            1, new ModelDestination(new ModelIdentifier("mixer", "active", 0)));

        defaultconnections[o++] = new ModelConnectionBlock(
            new ModelSource(
                new ModelIdentifier("eg", 0),
                ModelStandardTransform.DIRECTION_MAX2MIN,
                ModelStandardTransform.POLARITY_UNIPOLAR,
                ModelStandardTransform.TRANSFORM_LINEAR),
            -960, new ModelDestination(new ModelIdentifier("mixer", "gain")));

        defaultconnections[o++] = new ModelConnectionBlock(
            new ModelSource(
                new ModelIdentifier("noteon", "velocity"),
                ModelStandardTransform.DIRECTION_MAX2MIN,
                ModelStandardTransform.POLARITY_UNIPOLAR,
                ModelStandardTransform.TRANSFORM_CONCAVE),
            -960, new ModelDestination(new ModelIdentifier("mixer", "gain")));

        defaultconnections[o++] = new ModelConnectionBlock(
            new ModelSource(
                new ModelIdentifier("midi", "pitch"),
                ModelStandardTransform.DIRECTION_MIN2MAX,
                ModelStandardTransform.POLARITY_BIPOLAR,
                ModelStandardTransform.TRANSFORM_LINEAR),
            new ModelSource(new ModelIdentifier("midi_rpn", "0"),
                new ModelTransform() {
                    public double transform(double value) {
                        int v = (int) (value * 16384.0);
                        int msb = v >> 7;
                        int lsb = v & 127;
                        return msb * 100 + lsb;
                    }
                }),
            new ModelDestination(new ModelIdentifier("osc", "pitch")));

        defaultconnections[o++] = new ModelConnectionBlock(
            new ModelSource(
                new ModelIdentifier("noteon", "keynumber"),
                ModelStandardTransform.DIRECTION_MIN2MAX,
                ModelStandardTransform.POLARITY_UNIPOLAR,
                ModelStandardTransform.TRANSFORM_LINEAR),
            12800, new ModelDestination(new ModelIdentifier("osc", "pitch")));

        defaultconnections[o++] = new ModelConnectionBlock(
            new ModelSource(
                new ModelIdentifier("midi_cc", "7"),
                ModelStandardTransform.DIRECTION_MAX2MIN,
                ModelStandardTransform.POLARITY_UNIPOLAR,
                ModelStandardTransform.TRANSFORM_CONCAVE),
            -960, new ModelDestination(new ModelIdentifier("mixer", "gain")));

        defaultconnections[o++] = new ModelConnectionBlock(
            new ModelSource(
                new ModelIdentifier("midi_cc", "8"),
```

```
            ModelStandardTransform.DIRECTION_MIN2MAX,
            ModelStandardTransform.POLARITY_UNIPOLAR,
            ModelStandardTransform.TRANSFORM_LINEAR),
        1000, new ModelDestination(new ModelIdentifier("mixer", "balance")));

    defaultconnections[o++] = new ModelConnectionBlock(
        new ModelSource(
            new ModelIdentifier("midi_cc", "10"),
            ModelStandardTransform.DIRECTION_MIN2MAX,
            ModelStandardTransform.POLARITY_UNIPOLAR,
            ModelStandardTransform.TRANSFORM_LINEAR),
        1000, new ModelDestination(new ModelIdentifier("mixer", "pan")));

    defaultconnections[o++] = new ModelConnectionBlock(
        new ModelSource(
            new ModelIdentifier("midi_cc", "11"),
            ModelStandardTransform.DIRECTION_MAX2MIN,
            ModelStandardTransform.POLARITY_UNIPOLAR,
            ModelStandardTransform.TRANSFORM_CONCAVE),
        -960, new ModelDestination(new ModelIdentifier("mixer", "gain")));

    defaultconnections[o++] = new ModelConnectionBlock(
        new ModelSource(
            new ModelIdentifier("midi_cc", "91"),
            ModelStandardTransform.DIRECTION_MIN2MAX,
            ModelStandardTransform.POLARITY_UNIPOLAR,
            ModelStandardTransform.TRANSFORM_LINEAR),
        1000, new ModelDestination(new ModelIdentifier("mixer", "reverb")));

    defaultconnections[o++] = new ModelConnectionBlock(
        new ModelSource(
            new ModelIdentifier("midi_cc", "93"),
            ModelStandardTransform.DIRECTION_MIN2MAX,
            ModelStandardTransform.POLARITY_UNIPOLAR,
            ModelStandardTransform.TRANSFORM_LINEAR),
        1000, new ModelDestination(new ModelIdentifier("mixer", "chorus")));

    defaultconnections[o++] = new ModelConnectionBlock(
        new ModelSource(
            new ModelIdentifier("midi_cc", "71"),
            ModelStandardTransform.DIRECTION_MIN2MAX,
            ModelStandardTransform.POLARITY_BIPOLAR,
            ModelStandardTransform.TRANSFORM_LINEAR),
        200, new ModelDestination(new ModelIdentifier("filter", "q")));
    defaultconnections[o++] = new ModelConnectionBlock(
        new ModelSource(
            new ModelIdentifier("midi_cc", "74"),
            ModelStandardTransform.DIRECTION_MIN2MAX,
            ModelStandardTransform.POLARITY_BIPOLAR,
            ModelStandardTransform.TRANSFORM_LINEAR),
        9600, new ModelDestination(new ModelIdentifier("filter", "freq")));

    defaultconnections[o++] = new ModelConnectionBlock(
        new ModelSource(
            new ModelIdentifier("midi_cc", "72"),
            ModelStandardTransform.DIRECTION_MIN2MAX,
            ModelStandardTransform.POLARITY_BIPOLAR,
            ModelStandardTransform.TRANSFORM_LINEAR),
        6000, new ModelDestination(new ModelIdentifier("eg", "release2")));

    defaultconnections[o++] = new ModelConnectionBlock(
        new ModelSource(
```

```
            new ModelIdentifier("midi_cc", "73"),
            ModelStandardTransform.DIRECTION_MIN2MAX,
            ModelStandardTransform.POLARITY_BIPOLAR,
            ModelStandardTransform.TRANSFORM_LINEAR),
        2000, new ModelDestination(new ModelIdentifier("eg", "attack2")));

    defaultconnections[o++] = new ModelConnectionBlock(
        new ModelSource(
            new ModelIdentifier("midi_cc", "75"),
            ModelStandardTransform.DIRECTION_MIN2MAX,
            ModelStandardTransform.POLARITY_BIPOLAR,
            ModelStandardTransform.TRANSFORM_LINEAR),
        6000, new ModelDestination(new ModelIdentifier("eg", "decay2")));

    defaultconnections[o++] = new ModelConnectionBlock(
        new ModelSource(
            new ModelIdentifier("midi_cc", "67"),
            ModelStandardTransform.DIRECTION_MIN2MAX,
            ModelStandardTransform.POLARITY_UNIPOLAR,
            ModelStandardTransform.TRANSFORM_SWITCH),
        -50, new ModelDestination(ModelDestination.DESTINATION_GAIN));

    defaultconnections[o++] = new ModelConnectionBlock(
        new ModelSource(
            new ModelIdentifier("midi_cc", "67"),
            ModelStandardTransform.DIRECTION_MIN2MAX,
            ModelStandardTransform.POLARITY_UNIPOLAR,
            ModelStandardTransform.TRANSFORM_SWITCH),
        -2400, new ModelDestination(ModelDestination.DESTINATION_FILTER_FREQ));

    defaultconnections[o++] = new ModelConnectionBlock(
        new ModelSource(
            new ModelIdentifier("midi_rpn", "1"),
            ModelStandardTransform.DIRECTION_MIN2MAX,
            ModelStandardTransform.POLARITY_BIPOLAR,
            ModelStandardTransform.TRANSFORM_LINEAR),
        100, new ModelDestination(new ModelIdentifier("osc", "pitch")));

    defaultconnections[o++] = new ModelConnectionBlock(
        new ModelSource(
            new ModelIdentifier("midi_rpn", "2"),
            ModelStandardTransform.DIRECTION_MIN2MAX,
            ModelStandardTransform.POLARITY_BIPOLAR,
            ModelStandardTransform.TRANSFORM_LINEAR),
        12800, new ModelDestination(new ModelIdentifier("osc", "pitch")));

    defaultconnections[o++] = new ModelConnectionBlock(
        new ModelSource(
            new ModelIdentifier("master", "fine_tuning"),
            ModelStandardTransform.DIRECTION_MIN2MAX,
            ModelStandardTransform.POLARITY_BIPOLAR,
            ModelStandardTransform.TRANSFORM_LINEAR),
        100, new ModelDestination(new ModelIdentifier("osc", "pitch")));

    defaultconnections[o++] = new ModelConnectionBlock(
        new ModelSource(
            new ModelIdentifier("master", "coarse_tuning"),
            ModelStandardTransform.DIRECTION_MIN2MAX,
            ModelStandardTransform.POLARITY_BIPOLAR,
            ModelStandardTransform.TRANSFORM_LINEAR),
        12800, new ModelDestination(new ModelIdentifier("osc", "pitch")));

```

```
247         defaultconnections[o++] = new ModelConnectionBlock(13500,
248                 new ModelDestination(new ModelIdentifier("filter", "freq", 0)));
249 
250         defaultconnections[o++] = new ModelConnectionBlock(
251                 Float.NEGATIVE_INFINITY, new ModelDestination(
252                 new ModelIdentifier("eg", "delay", 0)));
253         defaultconnections[o++] = new ModelConnectionBlock(
254                 Float.NEGATIVE_INFINITY, new ModelDestination(
255                 new ModelIdentifier("eg", "attack", 0)));
256         defaultconnections[o++] = new ModelConnectionBlock(
257                 Float.NEGATIVE_INFINITY, new ModelDestination(
258                 new ModelIdentifier("eg", "hold", 0)));
259         defaultconnections[o++] = new ModelConnectionBlock(
260                 Float.NEGATIVE_INFINITY, new ModelDestination(
261                 new ModelIdentifier("eg", "decay", 0)));
262         defaultconnections[o++] = new ModelConnectionBlock(1000,
263                 new ModelDestination(new ModelIdentifier("eg", "sustain", 0)));
264         defaultconnections[o++] = new ModelConnectionBlock(
265                 Float.NEGATIVE_INFINITY, new ModelDestination(
266                 new ModelIdentifier("eg", "release", 0)));
267         defaultconnections[o++] = new ModelConnectionBlock(1200.0
268                 * Math.log(0.015) / Math.log(2), new ModelDestination(
269                 new ModelIdentifier("eg", "shutdown", 0))); // 15 msec default
270 
271         defaultconnections[o++] = new ModelConnectionBlock(
272                 Float.NEGATIVE_INFINITY, new ModelDestination(
273                 new ModelIdentifier("eg", "delay", 1)));
274         defaultconnections[o++] = new ModelConnectionBlock(
275                 Float.NEGATIVE_INFINITY, new ModelDestination(
276                 new ModelIdentifier("eg", "attack", 1)));
277         defaultconnections[o++] = new ModelConnectionBlock(
278                 Float.NEGATIVE_INFINITY, new ModelDestination(
279                 new ModelIdentifier("eg", "hold", 1)));
280         defaultconnections[o++] = new ModelConnectionBlock(
281                 Float.NEGATIVE_INFINITY, new ModelDestination(
282                 new ModelIdentifier("eg", "decay", 1)));
283         defaultconnections[o++] = new ModelConnectionBlock(1000,
284                 new ModelDestination(new ModelIdentifier("eg", "sustain", 1)));
285         defaultconnections[o++] = new ModelConnectionBlock(
286                 Float.NEGATIVE_INFINITY, new ModelDestination(
287                 new ModelIdentifier("eg", "release", 1)));
288 
289         defaultconnections[o++] = new ModelConnectionBlock(-8.51318,
290                 new ModelDestination(new ModelIdentifier("lfo", "freq", 0)));
291         defaultconnections[o++] = new ModelConnectionBlock(
292                 Float.NEGATIVE_INFINITY, new ModelDestination(
293                 new ModelIdentifier("lfo", "delay", 0)));
294         defaultconnections[o++] = new ModelConnectionBlock(-8.51318,
295                 new ModelDestination(new ModelIdentifier("lfo", "freq", 1)));
296         defaultconnections[o++] = new ModelConnectionBlock(
297                 Float.NEGATIVE_INFINITY, new ModelDestination(
298                 new ModelIdentifier("lfo", "delay", 1)));
299 
300     }
301     public int keyFrom = 0;
302     public int keyTo = 127;
303     public int velFrom = 0;
304     public int velTo = 127;
305     public int exclusiveClass = 0;
306     public boolean selfNonExclusive = false;
307     public boolean forcedVelocity = false;
308     public boolean forcedKeynumber = false;
```

```
    public ModelPerformer performer;
    public ModelConnectionBlock[] connections;
    public ModelOscillator[] oscillators;
    public Map<Integer, int[]> midi_rpn_connections = new HashMap<Integer, int[]>();
    public Map<Integer, int[]> midi_nrpn_connections = new HashMap<Integer, int[]>();
    public int[][] midi_ctrl_connections;
    public int[][] midi_connections;
    public int[] ctrl_connections;
    private List<Integer> ctrl_connections_list = new ArrayList<Integer>();

    private static class KeySortComparator implements Comparator<ModelSource> {

        public int compare(ModelSource o1, ModelSource o2) {
            return o1.getIdentifier().toString().compareTo(
                    o2.getIdentifier().toString());
        }
    }
    private static KeySortComparator keySortComparator = new KeySortComparator();

    private String extractKeys(ModelConnectionBlock conn) {
        StringBuffer sb = new StringBuffer();
        if (conn.getSources() != null) {
            sb.append("[");
            ModelSource[] srcs = conn.getSources();
            ModelSource[] srcs2 = new ModelSource[srcs.length];
            for (int i = 0; i < srcs.length; i++)
                srcs2[i] = srcs[i];
            Arrays.sort(srcs2, keySortComparator);
            for (int i = 0; i < srcs.length; i++) {
                sb.append(srcs[i].getIdentifier());
                sb.append(";");
            }
            sb.append("]");
        }
        sb.append(";");
        if (conn.getDestination() != null) {
            sb.append(conn.getDestination().getIdentifier());
        }
        sb.append(";");
        return sb.toString();
    }

    private void processSource(ModelSource src, int ix) {
        ModelIdentifier id = src.getIdentifier();
        String o = id.getObject();
        if (o.equals("midi_cc"))
            processMidiControlSource(src, ix);
        else if (o.equals("midi_rpn"))
            processMidiRpnSource(src, ix);
        else if (o.equals("midi_nrpn"))
            processMidiNrpnSource(src, ix);
        else if (o.equals("midi"))
            processMidiSource(src, ix);
        else if (o.equals("noteon"))
            processNoteOnSource(src, ix);
        else if (o.equals("osc"))
            return;
        else if (o.equals("mixer"))
            return;
        else
            ctrl_connections_list.add(ix);
    }
```

```
371
372     private void processMidiControlSource(ModelSource src, int ix) {
373         String v = src.getIdentifier().getVariable();
374         if (v == null)
375             return;
376         int c = Integer.parseInt(v);
377         if (midi_ctrl_connections[c] == null)
378             midi_ctrl_connections[c] = new int[]{ix};
379         else {
380             int[] olda = midi_ctrl_connections[c];
381             int[] newa = new int[olda.length + 1];
382             for (int i = 0; i < olda.length; i++)
383                 newa[i] = olda[i];
384             newa[newa.length - 1] = ix;
385             midi_ctrl_connections[c] = newa;
386         }
387     }
388
389     private void processNoteOnSource(ModelSource src, int ix) {
390         String v = src.getIdentifier().getVariable();
391         int c = -1;
392         if (v.equals("on"))
393             c = 3;
394         if (v.equals("keynumber"))
395             c = 4;
396         if (c == -1)
397             return;
398         if (midi_connections[c] == null)
399             midi_connections[c] = new int[]{ix};
400         else {
401             int[] olda = midi_connections[c];
402             int[] newa = new int[olda.length + 1];
403             for (int i = 0; i < olda.length; i++)
404                 newa[i] = olda[i];
405             newa[newa.length - 1] = ix;
406             midi_connections[c] = newa;
407         }
408     }
409
410     private void processMidiSource(ModelSource src, int ix) {
411         String v = src.getIdentifier().getVariable();
412         int c = -1;
413         if (v.equals("pitch"))
414             c = 0;
415         if (v.equals("channel_pressure"))
416             c = 1;
417         if (v.equals("poly_pressure"))
418             c = 2;
419         if (c == -1)
420             return;
421         if (midi_connections[c] == null)
422             midi_connections[c] = new int[]{ix};
423         else {
424             int[] olda = midi_connections[c];
425             int[] newa = new int[olda.length + 1];
426             for (int i = 0; i < olda.length; i++)
427                 newa[i] = olda[i];
428             newa[newa.length - 1] = ix;
429             midi_connections[c] = newa;
430         }
431     }
432
```

```
    private void processMidiRpnSource(ModelSource src, int ix) {
        String v = src.getIdentifier().getVariable();
        if (v == null)
            return;
        int c = Integer.parseInt(v);
        if (midi_rpn_connections.get(c) == null)
            midi_rpn_connections.put(c, new int[]{ix});
        else {
            int[] olda = midi_rpn_connections.get(c);
            int[] newa = new int[olda.length + 1];
            for (int i = 0; i < olda.length; i++)
                newa[i] = olda[i];
            newa[newa.length - 1] = ix;
            midi_rpn_connections.put(c, newa);
        }
    }

    private void processMidiNrpnSource(ModelSource src, int ix) {
        String v = src.getIdentifier().getVariable();
        if (v == null)
            return;
        int c = Integer.parseInt(v);
        if (midi_nrpn_connections.get(c) == null)
            midi_nrpn_connections.put(c, new int[]{ix});
        else {
            int[] olda = midi_nrpn_connections.get(c);
            int[] newa = new int[olda.length + 1];
            for (int i = 0; i < olda.length; i++)
                newa[i] = olda[i];
            newa[newa.length - 1] = ix;
            midi_nrpn_connections.put(c, newa);
        }
    }

    public SoftPerformer(ModelPerformer performer) {
        this.performer = performer;

        keyFrom = performer.getKeyFrom();
        keyTo = performer.getKeyTo();
        velFrom = performer.getVelFrom();
        velTo = performer.getVelTo();
        exclusiveClass = performer.getExclusiveClass();
        selfNonExclusive = performer.isSelfNonExclusive();

        Map<String, ModelConnectionBlock> connmap = new HashMap<String, ModelConnectionBlock>();

        List<ModelConnectionBlock> performer_connections = new ArrayList<ModelConnectionBlock>();
        performer_connections.addAll(performer.getConnectionBlocks());

        if (performer.isDefaultConnectionsEnabled()) {

            // Add modulation depth range (RPN 5) to the modulation wheel (cc#1)

            boolean isModulationWheelConectionFound = false;
            for (int j = 0; j < performer_connections.size(); j++) {
                ModelConnectionBlock connection = performer_connections.get(j);
                ModelSource[] sources = connection.getSources();
                ModelDestination dest = connection.getDestination();
                boolean isModulationWheelConection = false;
                if (dest != null && sources != null && sources.length > 1) {
                    for (int i = 0; i < sources.length; i++) {
                        // check if connection block has the source "modulation
```

```
                    // wheel cc#1"
                    if (sources[i].getIdentifier().getObject().equals(
                            "midi_cc")) {
                        if (sources[i].getIdentifier().getVariable()
                                .equals("1")) {
                            isModulationWheelConection = true;
                            isModulationWheelConectionFound = true;
                            break;
                        }
                    }
                }
            }
            if (isModulationWheelConection) {

                ModelConnectionBlock newconnection = new ModelConnectionBlock();
                newconnection.setSources(connection.getSources());
                newconnection.setDestination(connection.getDestination());
                newconnection.addSource(new ModelSource(
                        new ModelIdentifier("midi_rpn", "5")));
                newconnection.setScale(connection.getScale() * 256.0);
                performer_connections.set(j, newconnection);
            }
        }

        if (!isModulationWheelConectionFound) {
            ModelConnectionBlock conn = new ModelConnectionBlock(
                    new ModelSource(ModelSource.SOURCE_LFO1,
                    ModelStandardTransform.DIRECTION_MIN2MAX,
                    ModelStandardTransform.POLARITY_BIPOLAR,
                    ModelStandardTransform.TRANSFORM_LINEAR),
                    new ModelSource(new ModelIdentifier("midi_cc", "1", 0),
                    ModelStandardTransform.DIRECTION_MIN2MAX,
                    ModelStandardTransform.POLARITY_UNIPOLAR,
                    ModelStandardTransform.TRANSFORM_LINEAR),
                    50,
                    new ModelDestination(ModelDestination.DESTINATION_PITCH));
            conn.addSource(new ModelSource(new ModelIdentifier("midi_rpn",
                    "5")));
            conn.setScale(conn.getScale() * 256.0);
            performer_connections.add(conn);

        }

        // Let Aftertouch to behave just like modulation wheel (cc#1)
        boolean channel_pressure_set = false;
        boolean poly_pressure = false;
        ModelConnectionBlock mod_cc_1_connection = null;
        int mod_cc_1_connection_src_ix = 0;

        for (ModelConnectionBlock connection : performer_connections) {
            ModelSource[] sources = connection.getSources();
            ModelDestination dest = connection.getDestination();
            // if(dest != null && sources != null)
            if (dest != null && sources != null) {
                for (int i = 0; i < sources.length; i++) {
                    ModelIdentifier srcid = sources[i].getIdentifier();
                    // check if connection block has the source "modulation
                    // wheel cc#1"
                    if (srcid.getObject().equals("midi_cc")) {
                        if (srcid.getVariable().equals("1")) {
                            mod_cc_1_connection = connection;
                            mod_cc_1_connection_src_ix = i;
```

```
                    }
                }
                // check if channel or poly pressure are already
                // connected
                if (srcid.getObject().equals("midi")) {
                    if (srcid.getVariable().equals("channel_pressure"))
                        channel_pressure_set = true;
                    if (srcid.getVariable().equals("poly_pressure"))
                        poly_pressure = true;
                }
            }
        }

    }

    if (mod_cc_1_connection != null) {
        if (!channel_pressure_set) {
            ModelConnectionBlock mc = new ModelConnectionBlock();
            mc.setDestination(mod_cc_1_connection.getDestination());
            mc.setScale(mod_cc_1_connection.getScale());
            ModelSource[] src_list = mod_cc_1_connection.getSources();
            ModelSource[] src_list_new = new ModelSource[src_list.length];
            for (int i = 0; i < src_list_new.length; i++)
                src_list_new[i] = src_list[i];
            src_list_new[mod_cc_1_connection_src_ix] = new ModelSource(
                    new ModelIdentifier("midi", "channel_pressure"));
            mc.setSources(src_list_new);
            connmap.put(extractKeys(mc), mc);
        }
        if (!poly_pressure) {
            ModelConnectionBlock mc = new ModelConnectionBlock();
            mc.setDestination(mod_cc_1_connection.getDestination());
            mc.setScale(mod_cc_1_connection.getScale());
            ModelSource[] src_list = mod_cc_1_connection.getSources();
            ModelSource[] src_list_new = new ModelSource[src_list.length];
            for (int i = 0; i < src_list_new.length; i++)
                src_list_new[i] = src_list[i];
            src_list_new[mod_cc_1_connection_src_ix] = new ModelSource(
                    new ModelIdentifier("midi", "poly_pressure"));
            mc.setSources(src_list_new);
            connmap.put(extractKeys(mc), mc);
        }
    }

    // Enable Vibration Sound Controllers : 76, 77, 78
    ModelConnectionBlock found_vib_connection = null;
    for (ModelConnectionBlock connection : performer_connections) {
        ModelSource[] sources = connection.getSources();
        if (sources.length != 0
                && sources[0].getIdentifier().getObject().equals("lfo")) {
            if (connection.getDestination().getIdentifier().equals(
                    ModelDestination.DESTINATION_PITCH)) {
                if (found_vib_connection == null)
                    found_vib_connection = connection;
                else {
                    if (found_vib_connection.getSources().length > sources.length)
                        found_vib_connection = connection;
                    else if (found_vib_connection.getSources()[0]
                            .getIdentifier().getInstance() < 1) {
                        if (found_vib_connection.getSources()[0]
                                .getIdentifier().getInstance() >
                                sources[0].getIdentifier().getInstance()) {
```

```
                                found_vib_connection = connection;
                            }
                        }
                    }

                }
            }
        }

        int instance = 1;

        if (found_vib_connection != null) {
            instance = found_vib_connection.getSources()[0].getIdentifier()
                    .getInstance();
        }
        ModelConnectionBlock connection;

        connection = new ModelConnectionBlock(
            new ModelSource(new ModelIdentifier("midi_cc", "78"),
                ModelStandardTransform.DIRECTION_MIN2MAX,
                ModelStandardTransform.POLARITY_BIPOLAR,
                ModelStandardTransform.TRANSFORM_LINEAR),
            2000, new ModelDestination(
                new ModelIdentifier("lfo", "delay2", instance)));
        connmap.put(extractKeys(connection), connection);

        final double scale = found_vib_connection == null ? 0
                : found_vib_connection.getScale();
        connection = new ModelConnectionBlock(
            new ModelSource(new ModelIdentifier("lfo", instance)),
            new ModelSource(new ModelIdentifier("midi_cc", "77"),
                new ModelTransform() {
                    double s = scale;
                    public double transform(double value) {
                        value = value * 2 - 1;
                        value *= 600;
                        if (s == 0) {
                            return value;
                        } else if (s > 0) {
                            if (value < -s)
                                value = -s;
                            return value;
                        } else {
                            if (value < s)
                                value = -s;
                            return -value;
                        }
                    }
                }), new ModelDestination(ModelDestination.DESTINATION_PITCH));
        connmap.put(extractKeys(connection), connection);

        connection = new ModelConnectionBlock(
            new ModelSource(new ModelIdentifier("midi_cc", "76"),
                ModelStandardTransform.DIRECTION_MIN2MAX,
                ModelStandardTransform.POLARITY_BIPOLAR,
                ModelStandardTransform.TRANSFORM_LINEAR),
            2400, new ModelDestination(
                new ModelIdentifier("lfo", "freq", instance)));
        connmap.put(extractKeys(connection), connection);

    }

```

```
        // Add default connection blocks
        if (performer.isDefaultConnectionsEnabled())
            for (ModelConnectionBlock connection : defaultconnections)
                connmap.put(extractKeys(connection), connection);
        // Add connection blocks from modelperformer
        for (ModelConnectionBlock connection : performer_connections)
            connmap.put(extractKeys(connection), connection);
        // seperate connection blocks : Init time, Midi Time, Midi/Control Time,
        // Control Time
        List<ModelConnectionBlock> connections = new ArrayList<ModelConnectionBlock>();

        midi_ctrl_connections = new int[128][];
        for (int i = 0; i < midi_ctrl_connections.length; i++) {
            midi_ctrl_connections[i] = null;
        }
        midi_connections = new int[5][];
        for (int i = 0; i < midi_connections.length; i++) {
            midi_connections[i] = null;
        }

        int ix = 0;
        boolean mustBeOnTop = false;

        for (ModelConnectionBlock connection : connmap.values()) {
            if (connection.getDestination() != null) {
                ModelDestination dest = connection.getDestination();
                ModelIdentifier id = dest.getIdentifier();
                if (id.getObject().equals("noteon")) {
                    mustBeOnTop = true;
                    if (id.getVariable().equals("keynumber"))
                        forcedKeynumber = true;
                    if (id.getVariable().equals("velocity"))
                        forcedVelocity = true;
                }
            }
            if (mustBeOnTop) {
                connections.add(0, connection);
                mustBeOnTop = false;
            } else
                connections.add(connection);
        }

        for (ModelConnectionBlock connection : connections) {
            if (connection.getSources() != null) {
                ModelSource[] srcs = connection.getSources();
                for (int i = 0; i < srcs.length; i++) {
                    processSource(srcs[i], ix);
                }
            }
            ix++;
        }

        this.connections = new ModelConnectionBlock[connections.size()];
        connections.toArray(this.connections);

        this.ctrl_connections = new int[ctrl_connections_list.size()];

        for (int i = 0; i < this.ctrl_connections.length; i++)
            this.ctrl_connections[i] = ctrl_connections_list.get(i);

        oscillators = new ModelOscillator[performer.getOscillators().size()];
        performer.getOscillators().toArray(oscillators);
```

```
743
744         for (ModelConnectionBlock conn : connections) {
745             if (conn.getDestination() != null) {
746                 if (isUnnecessaryTransform(conn.getDestination().getTransform())) {
747                     conn.getDestination().setTransform(null);
748                 }
749             }
750             if (conn.getSources() != null) {
751                 for (ModelSource src : conn.getSources()) {
752                     if (isUnnecessaryTransform(src.getTransform())) {
753                         src.setTransform(null);
754                     }
755                 }
756             }
757         }
758
759     }
760
761     private static boolean isUnnecessaryTransform(ModelTransform transform) {
762         if (transform == null)
763             return false;
764         if (!(transform instanceof ModelStandardTransform))
765             return false;
766         ModelStandardTransform stransform = (ModelStandardTransform)transform;
767         if (stransform.getDirection() != ModelStandardTransform.DIRECTION_MIN2MAX)
768             return false;
769         if (stransform.getPolarity() != ModelStandardTransform.POLARITY_UNIPOLAR)
770             return false;
771         if (stransform.getTransform() != ModelStandardTransform.TRANSFORM_LINEAR)
772             return false;
773         return false;
774     }
775 }
```

104 com/sun/media/sound/SoftPointResampler.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * A resampler that uses 0-order (nearest-neighbor) interpolation.
 *
 * @author Karl Helgason
 */
public class SoftPointResampler extends SoftAbstractResampler {

    public int getPadding() {
        return 100;
    }

    public void interpolate(float[] in, float[] in_offset, float in_end,
            float[] startpitch, float pitchstep, float[] out, int[] out_offset,
            int out_end) {
        float pitch = startpitch[0];
        float ix = in_offset[0];
        int ox = out_offset[0];
        float ix_end = in_end;
        float ox_end = out_end;
        if (pitchstep == 0) {
            while (ix < ix_end && ox < ox_end) {
                out[ox++] = in[(int) ix];
                ix += pitch;
            }
        } else {
            while (ix < ix_end && ox < ox_end) {
                out[ox++] = in[(int) ix];
                ix += pitch;
                pitch += pitchstep;
            }
        }
        in_offset[0] = ix;
        out_offset[0] = ox;
        startpitch[0] = pitch;
```

```
61
62          }
63  }
```

105 com/sun/media/sound/SoftProcess.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * Control signal processor interface
 *
 * @author Karl Helgason
 */
public interface SoftProcess extends SoftControl {

    public void init(SoftSynthesizer synth);

    public double[] get(int instance, String name);

    public void processControlLogic();

    public void reset();
}
```

106 com/sun/media/sound/SoftProvider.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import javax.sound.midi.MidiDevice;
import javax.sound.midi.MidiDevice.Info;
import javax.sound.midi.spi.MidiDeviceProvider;

/**
 * Software synthesizer provider class.
 *
 * @author Karl Helgason
 */
public class SoftProvider extends MidiDeviceProvider {

    protected final static Info softinfo = SoftSynthesizer.info;
    private static Info[] softinfos = {softinfo};

    public MidiDevice.Info[] getDeviceInfo() {
        return softinfos;
    }

    public MidiDevice getDevice(MidiDevice.Info info) {
        if (info == softinfo) {
            return new SoftSynthesizer();
        }
        return null;
    }
}
```

107 com/sun/media/sound/SoftReceiver.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.util.TreeMap;

import javax.sound.midi.MidiDevice;
import javax.sound.midi.MidiMessage;
import javax.sound.midi.ShortMessage;

/**
 * Software synthesizer MIDI receiver class.
 *
 * @author Karl Helgason
 */
public class SoftReceiver implements MidiDeviceReceiver {

    protected boolean open = true;
    private Object control_mutex;
    private SoftSynthesizer synth;
    protected TreeMap<Long, Object> midimessages;
    protected SoftMainMixer mainmixer;

    public SoftReceiver(SoftSynthesizer synth) {
        this.control_mutex = synth.control_mutex;
        this.synth = synth;
        this.mainmixer = synth.getMainMixer();
        if (mainmixer != null)
            this.midimessages = mainmixer.midimessages;
    }

    public MidiDevice getMidiDevice() {
        return synth;
    }

    public void send(MidiMessage message, long timeStamp) {

        synchronized (control_mutex) {
```

```
            if (!open)
                throw new IllegalStateException("Receiver is not open");
        }

        if (timeStamp != -1) {
            synchronized (control_mutex) {
                mainmixer.activity();
                while (midimessages.get(timeStamp) != null)
                    timeStamp++;
                if (message instanceof ShortMessage
                        && (((ShortMessage)message).getChannel() > 0xF)) {
                    midimessages.put(timeStamp, message.clone());
                } else {
                    midimessages.put(timeStamp, message.getMessage());
                }
            }
        } else {
            mainmixer.processMessage(message);
        }
    }

    public void close() {
        synchronized (control_mutex) {
            open = false;
        }
        synth.removeReceiver(this);
    }
}
```

108 com/sun/media/sound/SoftResampler.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * Basic resampler interface.
 *
 * @author Karl Helgason
 */
public interface SoftResampler {

    public SoftResamplerStreamer openStreamer();
}
```

109 com/sun/media/sound/SoftResamplerStreamer.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.io.IOException;

/**
 * Resampler stream interface.
 *
 * @author Karl Helgason
 */
public interface SoftResamplerStreamer extends ModelOscillatorStream {

    public void open(ModelWavetable osc, float outputsamplerate)
            throws IOException;
}
```

110 com/sun/media/sound/SoftReverb.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.util.Arrays;

/**
 * Reverb effect based on allpass/comb filters. First audio is send to 8
 * parelled comb filters and then mixed together and then finally send thru 3
 * different allpass filters.
 *
 * @author Karl Helgason
 */
public class SoftReverb implements SoftAudioProcessor {

    private final static class Delay {

        private float[] delaybuffer;
        private int rovepos = 0;

        public Delay() {
            delaybuffer = null;
        }

        public void setDelay(int delay) {
            if (delay == 0)
                delaybuffer = null;
            else
                delaybuffer = new float[delay];
            rovepos = 0;
        }

        public void processReplace(float[] inout) {
            if (delaybuffer == null)
                return;
            int len = inout.length;
            int rnlen = delaybuffer.length;
            int rovepos = this.rovepos;
```

```
61
62              for (int i = 0; i < len; i++) {
63                  float x = inout[i];
64                  inout[i] = delaybuffer[rovepos];
65                  delaybuffer[rovepos] = x;
66                  if (++rovepos == rnlen)
67                      rovepos = 0;
68              }
69              this.rovepos = rovepos;
70          }
71      }
72
73      private final static class AllPass {
74
75          private final float[] delaybuffer;
76          private final int delaybuffersize;
77          private int rovepos = 0;
78          private float feedback;
79
80          public AllPass(int size) {
81              delaybuffer = new float[size];
82              delaybuffersize = size;
83          }
84
85          public void setFeedBack(float feedback) {
86              this.feedback = feedback;
87          }
88
89          public void processReplace(float inout[]) {
90              int len = inout.length;
91              int delaybuffersize = this.delaybuffersize;
92              int rovepos = this.rovepos;
93              for (int i = 0; i < len; i++) {
94                  float delayout = delaybuffer[rovepos];
95                  float input = inout[i];
96                  inout[i] = delayout - input;
97                  delaybuffer[rovepos] = input + delayout * feedback;
98                  if (++rovepos == delaybuffersize)
99                      rovepos = 0;
100             }
101             this.rovepos = rovepos;
102         }
103
104         public void processReplace(float in[], float out[]) {
105             int len = in.length;
106             int delaybuffersize = this.delaybuffersize;
107             int rovepos = this.rovepos;
108             for (int i = 0; i < len; i++) {
109                 float delayout = delaybuffer[rovepos];
110                 float input = in[i];
111                 out[i] = delayout - input;
112                 delaybuffer[rovepos] = input + delayout * feedback;
113                 if (++rovepos == delaybuffersize)
114                     rovepos = 0;
115             }
116             this.rovepos = rovepos;
117         }
118     }
119
120     private final static class Comb {
121
122         private final float[] delaybuffer;
```

```
123         private final int delaybuffersize;
124         private int rovepos = 0;
125         private float feedback;
126         private float filtertemp = 0;
127         private float filtercoeff1 = 0;
128         private float filtercoeff2 = 1;
129 
130         public Comb(int size) {
131             delaybuffer = new float[size];
132             delaybuffersize = size;
133         }
134 
135         public void setFeedBack(float feedback) {
136             this.feedback = feedback;
137             filtercoeff2 = (1 - filtercoeff1)* feedback;
138         }
139 
140         public void processMix(float in[], float out[]) {
141             int len = in.length;
142             int delaybuffersize = this.delaybuffersize;
143             int rovepos = this.rovepos;
144             float filtertemp = this.filtertemp;
145             float filtercoeff1 = this.filtercoeff1;
146             float filtercoeff2 = this.filtercoeff2;
147             for (int i = 0; i < len; i++) {
148                 float delayout = delaybuffer[rovepos];
149                 // One Pole Lowpass Filter
150                 filtertemp = (delayout * filtercoeff2)
151                         + (filtertemp * filtercoeff1);
152                 out[i] += delayout;
153                 delaybuffer[rovepos] = in[i] + filtertemp;
154                 if (++rovepos == delaybuffersize)
155                     rovepos = 0;
156             }
157             this.filtertemp  = filtertemp;
158             this.rovepos = rovepos;
159         }
160 
161         public void processReplace(float in[], float out[]) {
162             int len = in.length;
163             int delaybuffersize = this.delaybuffersize;
164             int rovepos = this.rovepos;
165             float filtertemp = this.filtertemp;
166             float filtercoeff1 = this.filtercoeff1;
167             float filtercoeff2 = this.filtercoeff2;
168             for (int i = 0; i < len; i++) {
169                 float delayout = delaybuffer[rovepos];
170                 // One Pole Lowpass Filter
171                 filtertemp = (delayout * filtercoeff2)
172                         + (filtertemp * filtercoeff1);
173                 out[i] = delayout;
174                 delaybuffer[rovepos] = in[i] + filtertemp;
175                 if (++rovepos == delaybuffersize)
176                     rovepos = 0;
177             }
178             this.filtertemp  = filtertemp;
179             this.rovepos = rovepos;
180         }
181 
182         public void setDamp(float val) {
183             filtercoeff1 = val;
184             filtercoeff2 = (1 - filtercoeff1)* feedback;
```

```
        }
    }
    private float roomsize;
    private float damp;
    private float gain = 1;
    private Delay delay;
    private Comb[] combL;
    private Comb[] combR;
    private AllPass[] allpassL;
    private AllPass[] allpassR;
    private float[] input;
    private float[] out;
    private float[] pre1;
    private float[] pre2;
    private float[] pre3;
    private boolean denormal_flip = false;
    private boolean mix = true;
    private SoftAudioBuffer inputA;
    private SoftAudioBuffer left;
    private SoftAudioBuffer right;
    private boolean dirty = true;
    private float dirty_roomsize;
    private float dirty_damp;
    private float dirty_predelay;
    private float dirty_gain;
    private float samplerate;
    private boolean light = true;

    public void init(float samplerate, float controlrate) {
        this.samplerate = samplerate;

        double freqscale = ((double) samplerate) / 44100.0;
        // freqscale = 1.0/ freqscale;

        int stereospread = 23;

        delay = new Delay();

        combL = new Comb[8];
        combR = new Comb[8];
        combL[0] = new Comb((int) (freqscale * (1116)));
        combR[0] = new Comb((int) (freqscale * (1116 + stereospread)));
        combL[1] = new Comb((int) (freqscale * (1188)));
        combR[1] = new Comb((int) (freqscale * (1188 + stereospread)));
        combL[2] = new Comb((int) (freqscale * (1277)));
        combR[2] = new Comb((int) (freqscale * (1277 + stereospread)));
        combL[3] = new Comb((int) (freqscale * (1356)));
        combR[3] = new Comb((int) (freqscale * (1356 + stereospread)));
        combL[4] = new Comb((int) (freqscale * (1422)));
        combR[4] = new Comb((int) (freqscale * (1422 + stereospread)));
        combL[5] = new Comb((int) (freqscale * (1491)));
        combR[5] = new Comb((int) (freqscale * (1491 + stereospread)));
        combL[6] = new Comb((int) (freqscale * (1557)));
        combR[6] = new Comb((int) (freqscale * (1557 + stereospread)));
        combL[7] = new Comb((int) (freqscale * (1617)));
        combR[7] = new Comb((int) (freqscale * (1617 + stereospread)));

        allpassL = new AllPass[4];
        allpassR = new AllPass[4];
        allpassL[0] = new AllPass((int) (freqscale * (556)));
        allpassR[0] = new AllPass((int) (freqscale * (556 + stereospread)));
        allpassL[1] = new AllPass((int) (freqscale * (441)));
```

```
        allpassR[1] = new AllPass((int) (freqscale * (441 + stereospread)));
        allpassL[2] = new AllPass((int) (freqscale * (341)));
        allpassR[2] = new AllPass((int) (freqscale * (341 + stereospread)));
        allpassL[3] = new AllPass((int) (freqscale * (225)));
        allpassR[3] = new AllPass((int) (freqscale * (225 + stereospread)));

        for (int i = 0; i < allpassL.length; i++) {
            allpassL[i].setFeedBack(0.5f);
            allpassR[i].setFeedBack(0.5f);
        }

        /* Init other settings */
        globalParameterControlChange(new int[]{0x01 * 128 + 0x01}, 0, 4);

    }

    public void setInput(int pin, SoftAudioBuffer input) {
        if (pin == 0)
            inputA = input;
    }

    public void setOutput(int pin, SoftAudioBuffer output) {
        if (pin == 0)
            left = output;
        if (pin == 1)
            right = output;
    }

    public void setMixMode(boolean mix) {
        this.mix = mix;
    }

    private boolean silent = true;

    public void processAudio() {
        boolean silent_input = this.inputA.isSilent();
        if(!silent_input)
            silent = false;
        if(silent)
        {
            if (!mix) {
                left.clear();
                right.clear();
            }
            return;
        }

        float[] inputA = this.inputA.array();
        float[] left = this.left.array();
        float[] right = this.right == null ? null : this.right.array();

        int numsamples = inputA.length;
        if (input == null || input.length < numsamples)
            input = new float[numsamples];

        float again = gain * 0.018f / 2;

        denormal_flip = !denormal_flip;
        if(denormal_flip)
            for (int i = 0; i < numsamples; i++)
                input[i] = inputA[i] * again + 1E-20f;
        else
```

```
            for (int i = 0; i < numsamples; i++)
                input[i] = inputA[i] * again - 1E-20f;

        delay.processReplace(input);

        if(light && (right != null))
        {
            if (pre1 == null || pre1.length < numsamples)
            {
                pre1 = new float[numsamples];
                pre2 = new float[numsamples];
                pre3 = new float[numsamples];
            }

            for (int i = 0; i < allpassL.length; i++)
                allpassL[i].processReplace(input);

            combL[0].processReplace(input, pre3);
            combL[1].processReplace(input, pre3);

            combL[2].processReplace(input, pre1);
            for (int i = 4; i < combL.length-2; i+=2)
                combL[i].processMix(input, pre1);

            combL[3].processReplace(input, pre2);;
            for (int i = 5; i < combL.length-2; i+=2)
                combL[i].processMix(input, pre2);

            if (!mix)
            {
                Arrays.fill(right, 0);
                Arrays.fill(left, 0);
            }
            for (int i = combR.length-2; i < combR.length; i++)
                combR[i].processMix(input, right);
            for (int i = combL.length-2; i < combL.length; i++)
                combL[i].processMix(input, left);

            for (int i = 0; i < numsamples; i++)
            {
                float p = pre1[i] - pre2[i];
                float m = pre3[i];
                left[i] += m + p;
                right[i] += m - p;
            }
        }
        else
        {
            if (out == null || out.length < numsamples)
                out = new float[numsamples];

            if (right != null) {
                if (!mix)
                    Arrays.fill(right, 0);
                allpassR[0].processReplace(input, out);
                for (int i = 1; i < allpassR.length; i++)
                    allpassR[i].processReplace(out);
                for (int i = 0; i < combR.length; i++)
                    combR[i].processMix(out, right);
            }

            if (!mix)
```

```
                Arrays.fill(left, 0);
            allpassL[0].processReplace(input, out);
            for (int i = 1; i < allpassL.length; i++)
                allpassL[i].processReplace(out);
            for (int i = 0; i < combL.length; i++)
                combL[i].processMix(out, left);
        }









        if (silent_input) {
            silent = true;
            for (int i = 0; i < numsamples; i++)
            {
                float v = left[i];
                if(v > 1E-10 || v < -1E-10)
                {
                    silent = false;
                    break;
                }
            }
        }

    }

    public void globalParameterControlChange(int[] slothpath, long param,
            long value) {
        if (slothpath.length == 1) {
            if (slothpath[0] == 0x01 * 128 + 0x01) {

                if (param == 0) {
                    if (value == 0) {
                        // Small Room A small size room with a length
                        // of 5m or so.
                        dirty_roomsize = (1.1f);
                        dirty_damp = (5000);
                        dirty_predelay = (0);
                        dirty_gain = (4);
                        dirty = true;
                    }
                    if (value == 1) {
                        // Medium Room A medium size room with a length
                        // of 10m or so.
                        dirty_roomsize = (1.3f);
                        dirty_damp = (5000);
                        dirty_predelay = (0);
                        dirty_gain = (3);
                        dirty = true;
                    }
                    if (value == 2) {
                        // Large Room A large size room suitable for
                        // live performances.
                        dirty_roomsize = (1.5f);
                        dirty_damp = (5000);
                        dirty_predelay = (0);
                        dirty_gain = (2);
                        dirty = true;
                    }
                    if (value == 3) {
```

```
                        // Medium Hall A medium size concert hall.
                        dirty_roomsize = (1.8f);
                        dirty_damp = (24000);
                        dirty_predelay = (0.02f);
                        dirty_gain = (1.5f);
                        dirty = true;
                    }
                    if (value == 4) {
                        // Large Hall A large size concert hall
                        // suitable for a full orchestra.
                        dirty_roomsize = (1.8f);
                        dirty_damp = (24000);
                        dirty_predelay = (0.03f);
                        dirty_gain = (1.5f);
                        dirty = true;
                    }
                    if (value == 8) {
                        // Plate A plate reverb simulation.
                        dirty_roomsize = (1.3f);
                        dirty_damp = (2500);
                        dirty_predelay = (0);
                        dirty_gain = (6);
                        dirty = true;
                    }
                } else if (param == 1) {
                    dirty_roomsize = ((float) (Math.exp((value - 40) * 0.025)));
                    dirty = true;
                }

            }
        }
    }

    public void processControlLogic() {
        if (dirty) {
            dirty = false;
            setRoomSize(dirty_roomsize);
            setDamp(dirty_damp);
            setPreDelay(dirty_predelay);
            setGain(dirty_gain);
        }
    }

    public void setRoomSize(float value) {
        roomsize = 1 - (0.17f / value);

        for (int i = 0; i < combL.length; i++) {
            combL[i].feedback = roomsize;
            combR[i].feedback = roomsize;
        }
    }

    public void setPreDelay(float value) {
        delay.setDelay((int)(value * samplerate));
    }

    public void setGain(float gain) {
        this.gain = gain;
    }

    public void setDamp(float value) {
        double x = (value / samplerate) * (2 * Math.PI);
```

```
495         double cx = 2 - Math.cos(x);
496         damp = (float)(cx - Math.sqrt(cx * cx - 1));
497         if (damp > 1)
498             damp = 1;
499         if (damp < 0)
500             damp = 0;
501 
502         // damp = value * 0.4f;
503         for (int i = 0; i < combL.length; i++) {
504             combL[i].setDamp(damp);
505             combR[i].setDamp(damp);
506         }
507 
508     }
509 
510     public void setLightMode(boolean light)
511     {
512         this.light = light;
513     }
514 }
```

111 com/sun/media/sound/SoftShortMessage.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import javax.sound.midi.InvalidMidiDataException;
import javax.sound.midi.ShortMessage;

/**
 * A short message class that support for than 16 midi channels.
 *
 * @author Karl Helgason
 */
public class SoftShortMessage extends ShortMessage {

    int channel = 0;

    public int getChannel() {
        return channel;
    }

    public void setMessage(int command, int channel, int data1, int data2)
            throws InvalidMidiDataException {
        this.channel = channel;
        super.setMessage(command, channel & 0xF, data1, data2);
    }

    public Object clone() {
        SoftShortMessage clone = new SoftShortMessage();
        try {
            clone.setMessage(getCommand(), getChannel(), getData1(), getData2());
        } catch (InvalidMidiDataException e) {
            throw new IllegalArgumentException(e);
        }
        return clone;
    }
}
```

112 com/sun/media/sound/SoftSincResampler.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

/**
 * Hann windowed sinc interpolation resampler with anti-alias filtering.
 *
 * Using 30 points for the interpolation.
 *
 * @author Karl Helgason
 */
public class SoftSincResampler extends SoftAbstractResampler {

    float[][][] sinc_table;
    int sinc_scale_size = 100;
    int sinc_table_fsize = 800;
    int sinc_table_size = 30;
    int sinc_table_center = sinc_table_size / 2;

    public SoftSincResampler() {
        super();
        sinc_table = new float[sinc_scale_size][sinc_table_fsize][];
        for (int s = 0; s < sinc_scale_size; s++) {
            float scale = (float) (1.0 / (1.0 + Math.pow(s, 1.1) / 10.0));
            for (int i = 0; i < sinc_table_fsize; i++) {
                sinc_table[s][i] = sincTable(sinc_table_size,
                        -i / ((float)sinc_table_fsize), scale);
            }
        }
    }

    // Normalized sinc function
    public static double sinc(double x) {
        return (x == 0.0) ? 1.0 : Math.sin(Math.PI * x) / (Math.PI * x);
    }

    // Generate hann window suitable for windowing sinc
    public static float[] wHanning(int size, float offset) {
```

```
        float[] window_table = new float[size];
        for (int k = 0; k < size; k++) {
            window_table[k] = (float)(-0.5
                    * Math.cos(2.0 * Math.PI * (double)(k + offset)
                        / (double) size) + 0.5);
        }
        return window_table;
    }

    // Generate sinc table
    public static float[] sincTable(int size, float offset, float scale) {
        int center = size / 2;
        float[] w = wHanning(size, offset);
        for (int k = 0; k < size; k++)
            w[k] *= sinc((-center + k + offset) * scale) * scale;
        return w;
    }

    public int getPadding() // must be at least half of sinc_table_size
    {
        return sinc_table_size / 2 + 2;
    }

    public void interpolate(float[] in, float[] in_offset, float in_end,
            float[] startpitch, float pitchstep, float[] out, int[] out_offset,
            int out_end) {
        float pitch = startpitch[0];
        float ix = in_offset[0];
        int ox = out_offset[0];
        float ix_end = in_end;
        int ox_end = out_end;
        int max_p = sinc_scale_size - 1;
        if (pitchstep == 0) {

            int p = (int) ((pitch - 1) * 10.0f);
            if (p < 0)
                p = 0;
            else if (p > max_p)
                p = max_p;
            float[][] sinc_table_f = this.sinc_table[p];
            while (ix < ix_end && ox < ox_end) {
                int iix = (int) ix;
                float[] sinc_table =
                        sinc_table_f[(int)((ix - iix) * sinc_table_fsize)];
                int xx = iix - sinc_table_center;
                float y = 0;
                for (int i = 0; i < sinc_table_size; i++, xx++)
                    y += in[xx] * sinc_table[i];
                out[ox++] = y;
                ix += pitch;
            }
        } else {
            while (ix < ix_end && ox < ox_end) {
                int iix = (int) ix;
                int p = (int) ((pitch - 1) * 10.0f);
                if (p < 0)
                    p = 0;
                else if (p > max_p)
                    p = max_p;
                float[][] sinc_table_f = this.sinc_table[p];

                float[] sinc_table =
```

```
                        sinc_table_f[(int)((ix - iix) * sinc_table_fsize)];
                int xx = iix - sinc_table_center;
                float y = 0;
                for (int i = 0; i < sinc_table_size; i++, xx++)
                    y += in[xx] * sinc_table[i];
                out[ox++] = y;

                ix += pitch;
                pitch += pitchstep;
            }
        }
        in_offset[0] = ix;
        out_offset[0] = ox;
        startpitch[0] = pitch;

    }
}
```

113 com/sun/media/sound/SoftSynthesizer.java

```
/*
 * Copyright 2008-2010 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */

package com.sun.media.sound;

import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.lang.ref.WeakReference;
import java.security.AccessController;
import java.security.PrivilegedAction;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.StringTokenizer;
import java.util.prefs.BackingStoreException;
import java.util.prefs.Preferences;

import javax.sound.midi.Instrument;
import javax.sound.midi.MidiChannel;
import javax.sound.midi.MidiDevice;
import javax.sound.midi.MidiSystem;
import javax.sound.midi.MidiUnavailableException;
import javax.sound.midi.Patch;
import javax.sound.midi.Receiver;
import javax.sound.midi.Soundbank;
import javax.sound.midi.Transmitter;
import javax.sound.midi.VoiceStatus;
import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
```

```
import javax.sound.sampled.LineUnavailableException;
import javax.sound.sampled.SourceDataLine;

/**
 * The software synthesizer class.
 *
 * @author Karl Helgason
 */
public class SoftSynthesizer implements AudioSynthesizer,
        ReferenceCountingDevice {

    protected static class WeakAudioStream extends InputStream
    {
        private volatile AudioInputStream stream;
        public SoftAudioPusher pusher = null;
        public AudioInputStream jitter_stream = null;
        public SourceDataLine sourceDataLine = null;
        public volatile long silent_samples = 0;
        private int framesize = 0;
        private WeakReference<AudioInputStream> weak_stream_link;
        private AudioFloatConverter converter;
        private float[] silentbuffer = null;
        private int samplesize;

        public void setInputStream(AudioInputStream stream)
        {
            this.stream = stream;
        }

        public int available() throws IOException {
            AudioInputStream local_stream = stream;
            if(local_stream != null)
                return local_stream.available();
            return 0;
        }

        public int read() throws IOException {
             byte[] b = new byte[1];
             if (read(b) == -1)
                  return -1;
             return b[0] & 0xFF;
        }

        public int read(byte[] b, int off, int len) throws IOException {
             AudioInputStream local_stream = stream;
             if(local_stream != null)
                 return local_stream.read(b, off, len);
             else
             {
                 int flen = len / samplesize;
                 if(silentbuffer == null || silentbuffer.length < flen)
                     silentbuffer = new float[flen];
                 converter.toByteArray(silentbuffer, flen, b, off);

                 silent_samples += (long)((len / framesize));

                 if(pusher != null)
                 if(weak_stream_link.get() == null)
                 {
                     Runnable runnable = new Runnable()
                     {
                         SoftAudioPusher _pusher = pusher;
```

```
                        AudioInputStream _jitter_stream = jitter_stream;
                        SourceDataLine _sourceDataLine = sourceDataLine;
                        public void run()
                        {
                            _pusher.stop();
                            if(_jitter_stream != null)
                               try {
                                   _jitter_stream.close();
                               } catch (IOException e) {
                                   e.printStackTrace();
                               }
                            if(_sourceDataLine != null)
                                _sourceDataLine.close();
                        }
                    };
                    pusher = null;
                    jitter_stream = null;
                    sourceDataLine = null;
                    new Thread(runnable).start();
                }
                return len;
            }
        }

        public WeakAudioStream(AudioInputStream stream) {
            this.stream = stream;
            weak_stream_link = new WeakReference<AudioInputStream>(stream);
            converter = AudioFloatConverter.getConverter(stream.getFormat());
            samplesize = stream.getFormat().getFrameSize() / stream.getFormat().getChannels();
            framesize = stream.getFormat().getFrameSize();
        }

        public AudioInputStream getAudioInputStream()
        {
            return new AudioInputStream(this, stream.getFormat(), AudioSystem.NOT_SPECIFIED);
        }

        public void close() throws IOException
        {
            AudioInputStream astream  = weak_stream_link.get();
            if(astream != null)
                astream.close();
        }
    }

    private static class Info extends MidiDevice.Info {
        public Info() {
            super(INFO_NAME, INFO_VENDOR, INFO_DESCRIPTION, INFO_VERSION);
        }
    }

    protected static final String INFO_NAME = "Gervill";
    protected static final String INFO_VENDOR = "OpenJDK";
    protected static final String INFO_DESCRIPTION = "Software␣MIDI␣Synthesizer";
    protected static final String INFO_VERSION = "1.0";
    protected final static MidiDevice.Info info = new Info();

    private static SourceDataLine testline = null;

    private static Soundbank defaultSoundBank = null;

    protected WeakAudioStream weakstream = null;
```

```
185
186     protected Object control_mutex = this;
187
188     protected int voiceIDCounter = 0;
189
190     // 0: default
191     // 1: DLS Voice Allocation
192     protected int voice_allocation_mode = 0;
193
194     protected boolean load_default_soundbank = false;
195     protected boolean reverb_light = true;
196     protected boolean reverb_on = true;
197     protected boolean chorus_on = true;
198     protected boolean agc_on = true;
199
200     protected SoftChannel[] channels;
201     protected SoftChannelProxy[] external_channels = null;
202
203     private boolean largemode = false;
204
205     // 0: GM Mode off (default)
206     // 1: GM Level 1
207     // 2: GM Level 2
208     private int gmmode = 0;
209
210     private int deviceid = 0;
211
212     private AudioFormat format = new AudioFormat(44100, 16, 2, true, false);
213
214     private SourceDataLine sourceDataLine = null;
215
216     private SoftAudioPusher pusher = null;
217     private AudioInputStream pusher_stream = null;
218
219     private float controlrate = 147f;
220
221     private boolean open = false;
222     private boolean implicitOpen = false;
223
224     private String resamplerType = "linear";
225     private SoftResampler resampler = new SoftLinearResampler();
226
227     private int number_of_midi_channels = 16;
228     private int maxpoly = 64;
229     private long latency = 200000; // 200 msec
230     private boolean jitter_correction = false;
231
232     private SoftMainMixer mainmixer;
233     private SoftVoice[] voices;
234
235     private Map<String, SoftTuning> tunings
236             = new HashMap<String, SoftTuning>();
237     private Map<String, SoftInstrument> inslist
238             = new HashMap<String, SoftInstrument>();
239     private Map<String, ModelInstrument> loadedlist
240             = new HashMap<String, ModelInstrument>();
241
242     private ArrayList<Receiver> recvslist = new ArrayList<Receiver>();
243
244     private void getBuffers(ModelInstrument instrument,
245             List<ModelByteBuffer> buffers) {
246         for (ModelPerformer performer : instrument.getPerformers()) {
```

```
            if (performer.getOscillators() != null) {
                for (ModelOscillator osc : performer.getOscillators()) {
                    if (osc instanceof ModelByteBufferWavetable) {
                        ModelByteBufferWavetable w = (ModelByteBufferWavetable)osc;
                        ModelByteBuffer buff = w.getBuffer();
                        if (buff != null)
                            buffers.add(buff);
                        buff = w.get8BitExtensionBuffer();
                        if (buff != null)
                            buffers.add(buff);
                    }
                }
            }
        }
    }

    private boolean loadSamples(List<ModelInstrument> instruments) {
        if (largemode)
            return true;
        List<ModelByteBuffer> buffers = new ArrayList<ModelByteBuffer>();
        for (ModelInstrument instrument : instruments)
            getBuffers(instrument, buffers);
        try {
            ModelByteBuffer.loadAll(buffers);
        } catch (IOException e) {
            return false;
        }
        return true;
    }

    private boolean loadInstruments(List<ModelInstrument> instruments) {
        if (!isOpen())
            return false;
        if (!loadSamples(instruments))
            return false;

        synchronized (control_mutex) {
            if (channels != null)
                for (SoftChannel c : channels)
                {
                    c.current_instrument = null;
                    c.current_director = null;
                }
            for (Instrument instrument : instruments) {
                String pat = patchToString(instrument.getPatch());
                SoftInstrument softins
                        = new SoftInstrument((ModelInstrument) instrument);
                inslist.put(pat, softins);
                loadedlist.put(pat, (ModelInstrument) instrument);
            }
        }

        return true;
    }

    private void processPropertyInfo(Map<String, Object> info) {
        AudioSynthesizerPropertyInfo[] items = getPropertyInfo(info);

        String resamplerType = (String)items[0].value;
        if (resamplerType.equalsIgnoreCase("point"))
        {
            this.resampler = new SoftPointResampler();
```

```
            this.resamplerType = "point";
        }
        else if (resamplerType.equalsIgnoreCase("linear"))
        {
            this.resampler = new SoftLinearResampler2();
            this.resamplerType = "linear";
        }
        else if (resamplerType.equalsIgnoreCase("linear1"))
        {
            this.resampler = new SoftLinearResampler();
            this.resamplerType = "linear1";
        }
        else if (resamplerType.equalsIgnoreCase("linear2"))
        {
            this.resampler = new SoftLinearResampler2();
            this.resamplerType = "linear2";
        }
        else if (resamplerType.equalsIgnoreCase("cubic"))
        {
            this.resampler = new SoftCubicResampler();
            this.resamplerType = "cubic";
        }
        else if (resamplerType.equalsIgnoreCase("lanczos"))
        {
            this.resampler = new SoftLanczosResampler();
            this.resamplerType = "lanczos";
        }
        else if (resamplerType.equalsIgnoreCase("sinc"))
        {
            this.resampler = new SoftSincResampler();
            this.resamplerType = "sinc";
        }

        setFormat((AudioFormat)items[2].value);
        controlrate = (Float)items[1].value;
        latency = (Long)items[3].value;
        deviceid = (Integer)items[4].value;
        maxpoly = (Integer)items[5].value;
        reverb_on = (Boolean)items[6].value;
        chorus_on = (Boolean)items[7].value;
        agc_on = (Boolean)items[8].value;
        largemode = (Boolean)items[9].value;
        number_of_midi_channels = (Integer)items[10].value;
        jitter_correction = (Boolean)items[11].value;
        reverb_light = (Boolean)items[12].value;
        load_default_soundbank = (Boolean)items[13].value;
    }

    private String patchToString(Patch patch) {
        if (patch instanceof ModelPatch && ((ModelPatch) patch).isPercussion())
            return "p." + patch.getProgram() + "." + patch.getBank();
        else
            return patch.getProgram() + "." + patch.getBank();
    }

    private void setFormat(AudioFormat format) {
        if (format.getChannels() > 2) {
            throw new IllegalArgumentException(
                    "Only mono and stereo audio supported.");
        }
        if (AudioFloatConverter.getConverter(format) == null)
            throw new IllegalArgumentException("Audio format not supported.");
```

```
        this.format = format;
    }

    protected void removeReceiver(Receiver recv) {
        boolean perform_close = false;
        synchronized (control_mutex) {
            if (recvslist.remove(recv)) {
                if (implicitOpen && recvslist.isEmpty())
                    perform_close = true;
            }
        }
        if (perform_close)
            close();
    }

    protected SoftMainMixer getMainMixer() {
        if (!isOpen())
            return null;
        return mainmixer;
    }

    protected SoftInstrument findInstrument(int program, int bank, int channel) {

        // Add support for GM2 banks 0x78 and 0x79
        // as specified in DLS 2.2 in Section 1.4.6
        // which allows using percussion and melodic instruments
        // on all channels
        if (bank >> 7 == 0x78 || bank >> 7 == 0x79) {
            SoftInstrument current_instrument
                    = inslist.get(program + "." + bank);
            if (current_instrument != null)
                return current_instrument;

            String p_plaf;
            if (bank >> 7 == 0x78)
                p_plaf = "p.";
            else
                p_plaf = "";

            // Instrument not found fallback to MSB:bank, LSB:0
            current_instrument = inslist.get(p_plaf + program + "."
                    + ((bank & 128) << 7));
            if (current_instrument != null)
                return current_instrument;
            // Instrument not found fallback to MSB:0, LSB:bank
            current_instrument = inslist.get(p_plaf + program + "."
                    + (bank & 128));
            if (current_instrument != null)
                return current_instrument;
            // Instrument not found fallback to MSB:0, LSB:0
            current_instrument = inslist.get(p_plaf + program + ".0");
            if (current_instrument != null)
                return current_instrument;
            // Instrument not found fallback to MSB:0, LSB:0, program=0
            current_instrument = inslist.get(p_plaf + program + "0.0");
            if (current_instrument != null)
                return current_instrument;
            return null;
        }

        // Channel 10 uses percussion instruments
        String p_plaf;
```

```
        if (channel == 9)
            p_plaf = "p.";
        else
            p_plaf = "";

        SoftInstrument current_instrument
                = inslist.get(p_plaf + program + "." + bank);
        if (current_instrument != null)
            return current_instrument;
        // Instrument not found fallback to MSB:0, LSB:0
        current_instrument = inslist.get(p_plaf + program + ".0");
        if (current_instrument != null)
            return current_instrument;
        // Instrument not found fallback to MSB:0, LSB:0, program=0
        current_instrument = inslist.get(p_plaf + "0.0");
        if (current_instrument != null)
            return current_instrument;
        return null;
    }

    protected int getVoiceAllocationMode() {
        return voice_allocation_mode;
    }

    protected int getGeneralMidiMode() {
        return gmmode;
    }

    protected void setGeneralMidiMode(int gmmode) {
        this.gmmode = gmmode;
    }

    protected int getDeviceID() {
        return deviceid;
    }

    protected float getControlRate() {
        return controlrate;
    }

    protected SoftVoice[] getVoices() {
        return voices;
    }

    protected SoftTuning getTuning(Patch patch) {
        String t_id = patchToString(patch);
        SoftTuning tuning = tunings.get(t_id);
        if (tuning == null) {
            tuning = new SoftTuning(patch);
            tunings.put(t_id, tuning);
        }
        return tuning;
    }

    public long getLatency() {
        synchronized (control_mutex) {
            return latency;
        }
    }

    public AudioFormat getFormat() {
        synchronized (control_mutex) {
```

```
            return format;
        }
    }

    public int getMaxPolyphony() {
        synchronized (control_mutex) {
            return maxpoly;
        }
    }

    public MidiChannel[] getChannels() {

        synchronized (control_mutex) {
            // if (external_channels == null) => the synthesizer is not open,
            // create 16 proxy channels
            // otherwise external_channels has the same length as channels array
            if (external_channels == null) {
                external_channels = new SoftChannelProxy[16];
                for (int i = 0; i < external_channels.length; i++)
                    external_channels[i] = new SoftChannelProxy();
            }
            MidiChannel[] ret;
            if (isOpen())
                ret = new MidiChannel[channels.length];
            else
                ret = new MidiChannel[16];
            for (int i = 0; i < ret.length; i++)
                ret[i] = external_channels[i];
            return ret;
        }
    }

    public VoiceStatus[] getVoiceStatus() {
        if (!isOpen()) {
            VoiceStatus[] tempVoiceStatusArray
                    = new VoiceStatus[getMaxPolyphony()];
            for (int i = 0; i < tempVoiceStatusArray.length; i++) {
                VoiceStatus b = new VoiceStatus();
                b.active = false;
                b.bank = 0;
                b.channel = 0;
                b.note = 0;
                b.program = 0;
                b.volume = 0;
                tempVoiceStatusArray[i] = b;
            }
            return tempVoiceStatusArray;
        }

        synchronized (control_mutex) {
            VoiceStatus[] tempVoiceStatusArray = new VoiceStatus[voices.length];
            for (int i = 0; i < voices.length; i++) {
                VoiceStatus a = voices[i];
                VoiceStatus b = new VoiceStatus();
                b.active = a.active;
                b.bank = a.bank;
                b.channel = a.channel;
                b.note = a.note;
                b.program = a.program;
                b.volume = a.volume;
                tempVoiceStatusArray[i] = b;
            }
```

```
            return tempVoiceStatusArray;
        }
    }

    public boolean isSoundbankSupported(Soundbank soundbank) {
        for (Instrument ins: soundbank.getInstruments())
            if (!(ins instanceof ModelInstrument))
                return false;
        return true;
    }

    public boolean loadInstrument(Instrument instrument) {
        if (instrument == null || (!(instrument instanceof ModelInstrument))) {
            throw new IllegalArgumentException("Unsupported instrument: " +
                    instrument);
        }
        List<ModelInstrument> instruments = new ArrayList<ModelInstrument>();
        instruments.add((ModelInstrument)instrument);
        return loadInstruments(instruments);
    }

    public void unloadInstrument(Instrument instrument) {
        if (instrument == null || (!(instrument instanceof ModelInstrument))) {
            throw new IllegalArgumentException("Unsupported instrument: " +
                    instrument);
        }
        if (!isOpen())
            return;

        String pat = patchToString(instrument.getPatch());
        synchronized (control_mutex) {
            for (SoftChannel c: channels)
                c.current_instrument = null;
            inslist.remove(pat);
            loadedlist.remove(pat);
            for (int i = 0; i < channels.length; i++) {
                channels[i].allSoundOff();
            }
        }
    }

    public boolean remapInstrument(Instrument from, Instrument to) {

        if (from == null)
            throw new NullPointerException();
        if (to == null)
            throw new NullPointerException();
        if (!(from instanceof ModelInstrument)) {
            throw new IllegalArgumentException("Unsupported instrument: " +
                    from.toString());
        }
        if (!(to instanceof ModelInstrument)) {
            throw new IllegalArgumentException("Unsupported instrument: " +
                    to.toString());
        }
        if (!isOpen())
            return false;

        synchronized (control_mutex) {
            if (!loadedlist.containsValue(to))
                throw new IllegalArgumentException("Instrument to is not loaded.");
            unloadInstrument(from);
```

```
619             ModelMappedInstrument mfrom = new ModelMappedInstrument(
620                     (ModelInstrument)to, from.getPatch());
621             return loadInstrument(mfrom);
622         }
623     }
624
625     public Soundbank getDefaultSoundbank() {
626         synchronized (SoftSynthesizer.class) {
627             if (defaultSoundBank != null)
628                 return defaultSoundBank;
629
630             List<PrivilegedAction<InputStream>> actions =
631                 new ArrayList<PrivilegedAction<InputStream>>();
632
633             actions.add(new PrivilegedAction<InputStream>() {
634                 public InputStream run() {
635                     File javahome = new File(System.getProperties()
636                             .getProperty("java.home"));
637                     File libaudio = new File(new File(javahome, "lib"), "audio");
638                     if (libaudio.exists()) {
639                         File foundfile = null;
640                         File[] files = libaudio.listFiles();
641                         if (files != null) {
642                             for (int i = 0; i < files.length; i++) {
643                                 File file = files[i];
644                                 if (file.isFile()) {
645                                     String lname = file.getName().toLowerCase();
646                                     if (lname.endsWith(".sf2")
647                                             || lname.endsWith(".dls")) {
648                                         if (foundfile == null
649                                                 || (file.length() > foundfile
650                                                         .length())) {
651                                             foundfile = file;
652                                         }
653                                     }
654                                 }
655                             }
656                         }
657                         if (foundfile != null) {
658                             try {
659                                 return new FileInputStream(foundfile);
660                             } catch (IOException e) {
661                             }
662                         }
663                     }
664                     return null;
665                 }
666             });
667
668             actions.add(new PrivilegedAction<InputStream>() {
669                 public InputStream run() {
670                     if (System.getProperties().getProperty("os.name")
671                             .startsWith("Windows")) {
672                         File gm_dls = new File(System.getenv("SystemRoot")
673                                 + "\\system32\\drivers\\gm.dls");
674                         if (gm_dls.exists()) {
675                             try {
676                                 return new FileInputStream(gm_dls);
677                             } catch (IOException e) {
678                             }
679                         }
680                     }
```

```
                return null;
            }
        });

        actions.add(new PrivilegedAction<InputStream>() {
            public InputStream run() {
                /*
                 * Try to load saved generated soundbank
                 */
                File userhome = new File(System.getProperty("user.home"),
                        ".gervill");
                File emg_soundbank_file = new File(userhome,
                        "soundbank-emg.sf2");
                if (emg_soundbank_file.exists()) {
                    try {
                        return new FileInputStream(emg_soundbank_file);
                    } catch (IOException e) {
                    }
                }
                return null;
            }
        });

        for (PrivilegedAction<InputStream> action : actions) {
            try {
                InputStream is = AccessController.doPrivileged(action);
                if(is == null) continue;
                Soundbank sbk;
                try {
                    sbk = MidiSystem.getSoundbank(new BufferedInputStream(is));
                } finally {
                    is.close();
                }
                if (sbk != null) {
                    defaultSoundBank = sbk;
                    return defaultSoundBank;
                }
            } catch (Exception e) {
            }
        }

        try {
            /*
             * Generate emergency soundbank
             */
            defaultSoundBank = EmergencySoundbank.createSoundbank();
        } catch (Exception e) {
        }

        if (defaultSoundBank != null) {
            /*
             * Save generated soundbank to disk for faster future use.
             */
            OutputStream out = AccessController
                    .doPrivileged(new PrivilegedAction<OutputStream>() {
                        public OutputStream run() {
                            try {
                                File userhome = new File(System
                                        .getProperty("user.home"),
                                        ".gervill");
                                if (!userhome.exists())
                                    userhome.mkdirs();
```

```
                                File emg_soundbank_file = new File(
                                        userhome, "soundbank-emg.sf2");
                                if (emg_soundbank_file.exists())
                                    return null;
                                return new FileOutputStream(
                                        emg_soundbank_file);
                            } catch (IOException e) {
                            } catch (SecurityException e) {
                            }
                            return null;
                        }
                    });
            if (out != null) {
                try {
                    ((SF2Soundbank) defaultSoundBank).save(out);
                    out.close();
                } catch (IOException e) {
                }
            }
        }
    }
    return defaultSoundBank;
}

public Instrument[] getAvailableInstruments() {
    Soundbank defsbk = getDefaultSoundbank();
    if (defsbk == null)
        return new Instrument[0];
    Instrument[] inslist_array = defsbk.getInstruments();
    Arrays.sort(inslist_array, new ModelInstrumentComparator());
    return inslist_array;
}

public Instrument[] getLoadedInstruments() {
    if (!isOpen())
        return new Instrument[0];

    synchronized (control_mutex) {
        ModelInstrument[] inslist_array =
                new ModelInstrument[loadedlist.values().size()];
        loadedlist.values().toArray(inslist_array);
        Arrays.sort(inslist_array, new ModelInstrumentComparator());
        return inslist_array;
    }
}

public boolean loadAllInstruments(Soundbank soundbank) {
    List<ModelInstrument> instruments = new ArrayList<ModelInstrument>();
    for (Instrument ins: soundbank.getInstruments()) {
        if (ins == null || !(ins instanceof ModelInstrument)) {
            throw new IllegalArgumentException(
                    "Unsupported instrument: " + ins);
        }
        instruments.add((ModelInstrument)ins);
    }
    return loadInstruments(instruments);
}

public void unloadAllInstruments(Soundbank soundbank) {
    if (soundbank == null || !isSoundbankSupported(soundbank))
        throw new IllegalArgumentException("Unsupported soundbank: " + soundbank);

```

```
805         if (!isOpen())
806             return;
807 
808         for (Instrument ins: soundbank.getInstruments()) {
809             if (ins instanceof ModelInstrument) {
810                 unloadInstrument(ins);
811             }
812         }
813     }
814 
815     public boolean loadInstruments(Soundbank soundbank, Patch[] patchList) {
816         List<ModelInstrument> instruments = new ArrayList<ModelInstrument>();
817         for (Patch patch: patchList) {
818             Instrument ins = soundbank.getInstrument(patch);
819             if (ins == null || !(ins instanceof ModelInstrument)) {
820                 throw new IllegalArgumentException(
821                         "Unsupported instrument: " + ins);
822             }
823             instruments.add((ModelInstrument)ins);
824         }
825         return loadInstruments(instruments);
826     }
827 
828     public void unloadInstruments(Soundbank soundbank, Patch[] patchList) {
829         if (soundbank == null || !isSoundbankSupported(soundbank))
830             throw new IllegalArgumentException("Unsupported soundbank: " + soundbank);
831 
832         if (!isOpen())
833             return;
834 
835         for (Patch pat: patchList) {
836             Instrument ins = soundbank.getInstrument(pat);
837             if (ins instanceof ModelInstrument) {
838                 unloadInstrument(ins);
839             }
840         }
841     }
842 
843     public MidiDevice.Info getDeviceInfo() {
844         return info;
845     }
846 
847     private Properties getStoredProperties() {
848         return AccessController
849                 .doPrivileged(new PrivilegedAction<Properties>() {
850                     public Properties run() {
851                         Properties p = new Properties();
852                         String notePath = "/com/sun/media/sound/softsynthesizer";
853                         try {
854                             Preferences prefroot = Preferences.userRoot();
855                             if (prefroot.nodeExists(notePath)) {
856                                 Preferences prefs = prefroot.node(notePath);
857                                 String[] prefs_keys = prefs.keys();
858                                 for (String prefs_key : prefs_keys) {
859                                     String val = prefs.get(prefs_key, null);
860                                     if (val != null)
861                                         p.setProperty(prefs_key, val);
862                                 }
863                             }
864                         } catch (BackingStoreException e) {
865                         } catch (SecurityException e) {
866                         }
```

```
                    return p;
                }
            });
    }

    public AudioSynthesizerPropertyInfo[] getPropertyInfo(Map<String, Object> info) {
        List<AudioSynthesizerPropertyInfo> list =
                new ArrayList<AudioSynthesizerPropertyInfo>();

        AudioSynthesizerPropertyInfo item;

        // If info != null or synthesizer is closed
        //   we return how the synthesizer will be set on next open
        // If info == null and synthesizer is open
        //   we return current synthesizer properties.
        boolean o = info == null && open;

        item = new AudioSynthesizerPropertyInfo("interpolation", o?resamplerType:"linear");
        item.choices = new String[]{"linear", "linear1", "linear2", "cubic",
                                    "lanczos", "sinc", "point"};
        item.description = "Interpolation method";
        list.add(item);

        item = new AudioSynthesizerPropertyInfo("control rate", o?controlrate:147f);
        item.description = "Control rate";
        list.add(item);

        item = new AudioSynthesizerPropertyInfo("format",
                o?format:new AudioFormat(44100, 16, 2, true, false));
        item.description = "Default audio format";
        list.add(item);

        item = new AudioSynthesizerPropertyInfo("latency", o?latency:120000L);
        item.description = "Default latency";
        list.add(item);

        item = new AudioSynthesizerPropertyInfo("device id", o?deviceid:0);
        item.description = "Device ID for SysEx Messages";
        list.add(item);

        item = new AudioSynthesizerPropertyInfo("max polyphony", o?maxpoly:64);
        item.description = "Maximum polyphony";
        list.add(item);

        item = new AudioSynthesizerPropertyInfo("reverb", o?reverb_on:true);
        item.description = "Turn reverb effect on or off";
        list.add(item);

        item = new AudioSynthesizerPropertyInfo("chorus", o?chorus_on:true);
        item.description = "Turn chorus effect on or off";
        list.add(item);

        item = new AudioSynthesizerPropertyInfo("auto gain control", o?agc_on:true);
        item.description = "Turn auto gain control on or off";
        list.add(item);

        item = new AudioSynthesizerPropertyInfo("large mode", o?largemode:false);
        item.description = "Turn large mode on or off.";
        list.add(item);

        item = new AudioSynthesizerPropertyInfo("midi channels", o?channels.length:16);
        item.description = "Number of midi channels.";
```

```
        list.add(item);

        item = new AudioSynthesizerPropertyInfo("jitter correction", o?jitter_correction:true);
        item.description = "Turn jitter correction on or off.";
        list.add(item);

        item = new AudioSynthesizerPropertyInfo("light reverb", o?reverb_light:true);
        item.description = "Turn light reverb mode on or off";
        list.add(item);

        item = new AudioSynthesizerPropertyInfo("load default soundbank", o?
           load_default_soundbank:true);
        item.description = "Enabled/disable loading default soundbank";
        list.add(item);

        AudioSynthesizerPropertyInfo[] items;
        items = list.toArray(new AudioSynthesizerPropertyInfo[list.size()]);

        Properties storedProperties = getStoredProperties();

        for (AudioSynthesizerPropertyInfo item2 : items) {
            Object v = (info == null) ? null : info.get(item2.name);
            v = (v != null) ? v : storedProperties.getProperty(item2.name);
            if (v != null) {
                Class c = (item2.valueClass);
                if (c.isInstance(v))
                    item2.value = v;
                else if (v instanceof String) {
                    String s = (String) v;
                    if (c == Boolean.class) {
                        if (s.equalsIgnoreCase("true"))
                            item2.value = Boolean.TRUE;
                        if (s.equalsIgnoreCase("false"))
                            item2.value = Boolean.FALSE;
                    } else if (c == AudioFormat.class) {
                        int channels = 2;
                        boolean signed = true;
                        boolean bigendian = false;
                        int bits = 16;
                        float sampleRate = 44100f;
                        try {
                            StringTokenizer st = new StringTokenizer(s, ", ");
                            String prevToken = "";
                            while (st.hasMoreTokens()) {
                                String token = st.nextToken().toLowerCase();
                                if (token.equals("mono"))
                                    channels = 1;
                                if (token.startsWith("channel"))
                                    channels = Integer.parseInt(prevToken);
                                if (token.contains("unsigned"))
                                    signed = false;
                                if (token.equals("big-endian"))
                                    bigendian = true;
                                if (token.equals("bit"))
                                    bits = Integer.parseInt(prevToken);
                                if (token.equals("hz"))
                                    sampleRate = Float.parseFloat(prevToken);
                                prevToken = token;
                            }
                            item2.value = new AudioFormat(sampleRate, bits,
                                    channels, signed, bigendian);
                        } catch (NumberFormatException e) {
```

```
                    }

                } else
                    try {
                        if (c == Byte.class)
                            item2.value = Byte.valueOf(s);
                        else if (c == Short.class)
                            item2.value = Short.valueOf(s);
                        else if (c == Integer.class)
                            item2.value = Integer.valueOf(s);
                        else if (c == Long.class)
                            item2.value = Long.valueOf(s);
                        else if (c == Float.class)
                            item2.value = Float.valueOf(s);
                        else if (c == Double.class)
                            item2.value = Double.valueOf(s);
                    } catch (NumberFormatException e) {
                    }
            } else if (v instanceof Number) {
                Number n = (Number) v;
                if (c == Byte.class)
                    item2.value = Byte.valueOf(n.byteValue());
                if (c == Short.class)
                    item2.value = Short.valueOf(n.shortValue());
                if (c == Integer.class)
                    item2.value = Integer.valueOf(n.intValue());
                if (c == Long.class)
                    item2.value = Long.valueOf(n.longValue());
                if (c == Float.class)
                    item2.value = Float.valueOf(n.floatValue());
                if (c == Double.class)
                    item2.value = Double.valueOf(n.doubleValue());
            }
        }
    }

    return items;
}

public void open() throws MidiUnavailableException {
    if (isOpen()) {
        synchronized (control_mutex) {
            implicitOpen = false;
        }
        return;
    }
    open(null, null);
}

public void open(SourceDataLine line, Map<String, Object> info) throws
    MidiUnavailableException {
    if (isOpen()) {
        synchronized (control_mutex) {
            implicitOpen = false;
        }
        return;
    }
    synchronized (control_mutex) {
        Throwable causeException = null;
        try {
            if (line != null) {
                // can throw IllegalArgumentException
```

```
            setFormat(line.getFormat());
        }

        AudioInputStream ais = openStream(getFormat(), info);

        weakstream = new WeakAudioStream(ais);
        ais = weakstream.getAudioInputStream();

        if (line == null)
        {
            if (testline != null) {
                line = testline;
            } else {
                // can throw LineUnavailableException,
                // IllegalArgumentException, SecurityException
                line = AudioSystem.getSourceDataLine(getFormat());
            }
        }

        double latency = this.latency;

        if (!line.isOpen()) {
            int bufferSize = getFormat().getFrameSize()
                * (int)(getFormat().getFrameRate() * (latency/1000000f));
            // can throw LineUnavailableException,
            // IllegalArgumentException, SecurityException
            line.open(getFormat(), bufferSize);

            // Remember that we opened that line
            // so we can close again in SoftSynthesizer.close()
            sourceDataLine = line;
        }
        if (!line.isActive())
            line.start();

        int controlbuffersize = 512;
        try {
            controlbuffersize = ais.available();
        } catch (IOException e) {
        }

        // Tell mixer not fill read buffers fully.
        // This lowers latency, and tells DataPusher
        // to read in smaller amounts.
        //mainmixer.readfully = false;
        //pusher = new DataPusher(line, ais);

        int buffersize = line.getBufferSize();
        buffersize -= buffersize % controlbuffersize;

        if (buffersize < 3 * controlbuffersize)
            buffersize = 3 * controlbuffersize;

        if (jitter_correction) {
            ais = new SoftJitterCorrector(ais, buffersize,
                    controlbuffersize);
            if(weakstream != null)
                weakstream.jitter_stream = ais;
        }
        pusher = new SoftAudioPusher(line, ais, controlbuffersize);
        pusher_stream = ais;
        pusher.start();
```

```
            if(weakstream != null)
            {
                weakstream.pusher = pusher;
                weakstream.sourceDataLine = sourceDataLine;
            }

        } catch (LineUnavailableException e) {
            causeException = e;
        } catch (IllegalArgumentException e) {
            causeException = e;
        } catch (SecurityException e) {
            causeException = e;
        }

        if (causeException != null) {
            if (isOpen())
                close();
            // am: need MidiUnavailableException(Throwable) ctor!
            MidiUnavailableException ex = new MidiUnavailableException(
                    "Can not open line");
            ex.initCause(causeException);
            throw ex;
        }

    }
}

public AudioInputStream openStream(AudioFormat targetFormat,
        Map<String, Object> info) throws MidiUnavailableException {

    if (isOpen())
        throw new MidiUnavailableException("Synthesizer is already open");

    synchronized (control_mutex) {

        gmmode = 0;
        voice_allocation_mode = 0;

        processPropertyInfo(info);

        open = true;
        implicitOpen = false;

        if (targetFormat != null)
            setFormat(targetFormat);

        if (load_default_soundbank)
        {
            Soundbank defbank = getDefaultSoundbank();
            if (defbank != null) {
                loadAllInstruments(defbank);
            }
        }

        voices = new SoftVoice[maxpoly];
        for (int i = 0; i < maxpoly; i++)
            voices[i] = new SoftVoice(this);

        mainmixer = new SoftMainMixer(this);

        channels = new SoftChannel[number_of_midi_channels];
```

```
            for (int i = 0; i < channels.length; i++)
                channels[i] = new SoftChannel(this, i);

            if (external_channels == null) {
                // Always create external_channels array
                // with 16 or more channels
                // so getChannels works correctly
                // when the synhtesizer is closed.
                if (channels.length < 16)
                    external_channels = new SoftChannelProxy[16];
                else
                    external_channels = new SoftChannelProxy[channels.length];
                for (int i = 0; i < external_channels.length; i++)
                    external_channels[i] = new SoftChannelProxy();
            } else {
                // We must resize external_channels array
                // but we must also copy the old SoftChannelProxy
                // into the new one
                if (channels.length > external_channels.length) {
                    SoftChannelProxy[] new_external_channels
                            = new SoftChannelProxy[channels.length];
                    for (int i = 0; i < external_channels.length; i++)
                        new_external_channels[i] = external_channels[i];
                    for (int i = external_channels.length;
                            i < new_external_channels.length; i++) {
                        new_external_channels[i] = new SoftChannelProxy();
                    }
                }
            }

            for (int i = 0; i < channels.length; i++)
                external_channels[i].setChannel(channels[i]);

            for (SoftVoice voice: getVoices())
                voice.resampler = resampler.openStreamer();

            for (Receiver recv: getReceivers()) {
                SoftReceiver srecv = ((SoftReceiver)recv);
                srecv.open = open;
                srecv.mainmixer = mainmixer;
                srecv.midimessages = mainmixer.midimessages;
            }

            return mainmixer.getInputStream();
        }
    }

    public void close() {

        if (!isOpen())
            return;

        SoftAudioPusher pusher_to_be_closed = null;
        AudioInputStream pusher_stream_to_be_closed = null;
        synchronized (control_mutex) {
            if (pusher != null) {
                pusher_to_be_closed = pusher;
                pusher_stream_to_be_closed = pusher_stream;
                pusher = null;
                pusher_stream = null;
            }
        }
```

```
        if (pusher_to_be_closed != null) {
            // Pusher must not be closed synchronized against control_mutex,
            // this may result in synchronized conflict between pusher
            // and current thread.
            pusher_to_be_closed.stop();

            try {
                pusher_stream_to_be_closed.close();
            } catch (IOException e) {
                //e.printStackTrace();
            }
        }

        synchronized (control_mutex) {

            if (mainmixer != null)
                mainmixer.close();
            open = false;
            implicitOpen = false;
            mainmixer = null;
            voices = null;
            channels = null;

            if (external_channels != null)
                for (int i = 0; i < external_channels.length; i++)
                    external_channels[i].setChannel(null);

            if (sourceDataLine != null) {
                sourceDataLine.close();
                sourceDataLine = null;
            }

            inslist.clear();
            loadedlist.clear();
            tunings.clear();

            while (recvslist.size() != 0)
                recvslist.get(recvslist.size() - 1).close();

        }
    }

    public boolean isOpen() {
        synchronized (control_mutex) {
            return open;
        }
    }

    public long getMicrosecondPosition() {

        if (!isOpen())
            return 0;

        synchronized (control_mutex) {
            return mainmixer.getMicrosecondPosition();
        }
    }

    public int getMaxReceivers() {
        return -1;
    }
```

```
1299
1300        public int getMaxTransmitters() {
1301            return 0;
1302        }
1303
1304        public Receiver getReceiver() throws MidiUnavailableException {
1305
1306            synchronized (control_mutex) {
1307                SoftReceiver receiver = new SoftReceiver(this);
1308                receiver.open = open;
1309                recvslist.add(receiver);
1310                return receiver;
1311            }
1312        }
1313
1314        public List<Receiver> getReceivers() {
1315
1316            synchronized (control_mutex) {
1317                ArrayList<Receiver> recvs = new ArrayList<Receiver>();
1318                recvs.addAll(recvslist);
1319                return recvs;
1320            }
1321        }
1322
1323        public Transmitter getTransmitter() throws MidiUnavailableException {
1324
1325            throw new MidiUnavailableException("No transmitter available");
1326        }
1327
1328        public List<Transmitter> getTransmitters() {
1329
1330            return new ArrayList<Transmitter>();
1331        }
1332
1333        public Receiver getReceiverReferenceCounting()
1334                throws MidiUnavailableException {
1335
1336            if (!isOpen()) {
1337                open();
1338                synchronized (control_mutex) {
1339                    implicitOpen = true;
1340                }
1341            }
1342
1343            return getReceiver();
1344        }
1345
1346        public Transmitter getTransmitterReferenceCounting()
1347                throws MidiUnavailableException {
1348
1349            throw new MidiUnavailableException("No transmitter available");
1350        }
1351    }
```

114 com/sun/media/sound/SoftTuning.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.io.UnsupportedEncodingException;

import javax.sound.midi.Patch;

/**
 * A tuning program container, for use with MIDI Tuning.
 * See: http://www.midi.org
 *
 * @author Karl Helgason
 */
public class SoftTuning {

    private String name = null;
    private double[] tuning = new double[128];
    private Patch patch = null;

    public SoftTuning() {
        name = "12-TET";
        for (int i = 0; i < tuning.length; i++)
            tuning[i] = i * 100;
    }

    public SoftTuning(byte[] data) {
        for (int i = 0; i < tuning.length; i++)
            tuning[i] = i * 100;
        load(data);
    }

    public SoftTuning(Patch patch) {
        this.patch = patch;
        name = "12-TET";
        for (int i = 0; i < tuning.length; i++)
            tuning[i] = i * 100;
    }
```

```
61
62      public SoftTuning(Patch patch, byte[] data) {
63          this.patch = patch;
64          for (int i = 0; i < tuning.length; i++)
65              tuning[i] = i * 100;
66          load(data);
67      }
68
69      private boolean checksumOK(byte[] data) {
70          int x = data[1] & 0xFF;
71          for (int i = 2; i < data.length - 2; i++)
72              x = x ^ (data[i] & 0xFF);
73          return (data[data.length - 2] & 0xFF) == (x & 127);
74      }
75
76      /*
77      private boolean checksumOK2(byte[] data) {
78          int x = data[1] & 0xFF; // 7E
79          x = x ^ (data[2] & 0xFF); // <device ID>
80          x = x ^ (data[4] & 0xFF); // nn
81          x = x ^ (data[5] & 0xFF); // tt
82          for (int i = 22; i < data.length - 2; i++)
83              x = x ^ (data[i] & 0xFF);
84          return (data[data.length - 2] & 0xFF) == (x & 127);
85      }
86       */
87      public void load(byte[] data) {
88          // Universal Non-Real-Time / Real-Time SysEx
89          if ((data[1] & 0xFF) == 0x7E || (data[1] & 0xFF) == 0x7F) {
90              int subid1 = data[3] & 0xFF;
91              switch (subid1) {
92              case 0x08: // MIDI Tuning Standard
93                  int subid2 = data[4] & 0xFF;
94                  switch (subid2) {
95                  case 0x01: // BULK TUNING DUMP (NON-REAL-TIME)
96                  {
97                      // http://www.midi.org/about-midi/tuning.shtml
98                      //if (!checksumOK2(data))
99                      //  break;
100                     try {
101                         name = new String(data, 6, 16, "ascii");
102                     } catch (UnsupportedEncodingException e) {
103                         name = null;
104                     }
105                     int r = 22;
106                     for (int i = 0; i < 128; i++) {
107                         int xx = data[r++] & 0xFF;
108                         int yy = data[r++] & 0xFF;
109                         int zz = data[r++] & 0xFF;
110                         if (!(xx == 127 && yy == 127 && zz == 127))
111                             tuning[i] = 100.0 *
112                                     (((xx * 16384) + (yy * 128) + zz) / 16384.0);
113                     }
114                     break;
115                 }
116                 case 0x02: // SINGLE NOTE TUNING CHANGE (REAL-TIME)
117                 {
118                     // http://www.midi.org/about-midi/tuning.shtml
119                     int ll = data[6] & 0xFF;
120                     int r = 7;
121                     for (int i = 0; i < ll; i++) {
122                         int kk = data[r++] & 0xFF;
```

```
                int xx = data[r++] & 0xFF;
                int yy = data[r++] & 0xFF;
                int zz = data[r++] & 0xFF;
                if (!(xx == 127 && yy == 127 && zz == 127))
                    tuning[kk] = 100.0*(((xx*16384) + (yy*128) + zz)/16384.0);
            }
            break;
        }
        case 0x04: // KEY-BASED TUNING DUMP (NON-REAL-TIME)
        {
            // http://www.midi.org/about-midi/tuning_extens.shtml
            if (!checksumOK(data))
                break;
            try {
                name = new String(data, 7, 16, "ascii");
            } catch (UnsupportedEncodingException e) {
                name = null;
            }
            int r = 23;
            for (int i = 0; i < 128; i++) {
                int xx = data[r++] & 0xFF;
                int yy = data[r++] & 0xFF;
                int zz = data[r++] & 0xFF;
                if (!(xx == 127 && yy == 127 && zz == 127))
                    tuning[i] = 100.0*(((xx*16384) + (yy*128) + zz)/16384.0);
            }
            break;
        }
        case 0x05: // SCALE/OCTAVE TUNING DUMP, 1 byte format
                   // (NON-REAL-TIME)
        {
            // http://www.midi.org/about-midi/tuning_extens.shtml
            if (!checksumOK(data))
                break;
            try {
                name = new String(data, 7, 16, "ascii");
            } catch (UnsupportedEncodingException e) {
                name = null;
            }
            int[] octave_tuning = new int[12];
            for (int i = 0; i < 12; i++)
                octave_tuning[i] = (data[i + 23] & 0xFF) - 64;
            for (int i = 0; i < tuning.length; i++)
                tuning[i] = i * 100 + octave_tuning[i % 12];
            break;
        }
        case 0x06: // SCALE/OCTAVE TUNING DUMP, 2 byte format
                   // (NON-REAL-TIME)
        {
            // http://www.midi.org/about-midi/tuning_extens.shtml
            if (!checksumOK(data))
                break;
            try {
                name = new String(data, 7, 16, "ascii");
            } catch (UnsupportedEncodingException e) {
                name = null;
            }
            double[] octave_tuning = new double[12];
            for (int i = 0; i < 12; i++) {
                int v = (data[i * 2 + 23] & 0xFF) * 128
                        + (data[i * 2 + 24] & 0xFF);
                octave_tuning[i] = (v / 8192.0 - 1) * 100.0;
```

```
                    }
                    for (int i = 0; i < tuning.length; i++)
                        tuning[i] = i * 100 + octave_tuning[i % 12];
                    break;
                }
                case 0x07: // SINGLE NOTE TUNING CHANGE (NON
                           // REAL-TIME/REAL-TIME) (BANK)
                    // http://www.midi.org/about-midi/tuning_extens.shtml
                    int ll = data[7] & 0xFF;
                    int r = 8;
                    for (int i = 0; i < ll; i++) {
                        int kk = data[r++] & 0xFF;
                        int xx = data[r++] & 0xFF;
                        int yy = data[r++] & 0xFF;
                        int zz = data[r++] & 0xFF;
                        if (!(xx == 127 && yy == 127 && zz == 127))
                            tuning[kk] = 100.0
                                    * (((xx*16384) + (yy*128) + zz) / 16384.0);
                    }
                    break;
                case 0x08: // scale/octave tuning 1-byte form (Non
                           // Real-Time/REAL-TIME)
                {
                    // http://www.midi.org/about-midi/tuning-scale.shtml
                    int[] octave_tuning = new int[12];
                    for (int i = 0; i < 12; i++)
                        octave_tuning[i] = (data[i + 8] & 0xFF) - 64;
                    for (int i = 0; i < tuning.length; i++)
                        tuning[i] = i * 100 + octave_tuning[i % 12];
                    break;
                }
                case 0x09: // scale/octave tuning 2-byte form (Non
                           // Real-Time/REAL-TIME)
                {
                    // http://www.midi.org/about-midi/tuning-scale.shtml
                    double[] octave_tuning = new double[12];
                    for (int i = 0; i < 12; i++) {
                        int v = (data[i * 2 + 8] & 0xFF) * 128
                                + (data[i * 2 + 9] & 0xFF);
                        octave_tuning[i] = (v / 8192.0 - 1) * 100.0;
                    }
                    for (int i = 0; i < tuning.length; i++)
                        tuning[i] = i * 100 + octave_tuning[i % 12];
                    break;
                }
                default:
                    break;
                }
            }
        }
    }

    public double[] getTuning() {
        return tuning;
    }

    public double getTuning(int noteNumber) {
        return tuning[noteNumber];
    }

    public Patch getPatch() {
        return patch;
```

```
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

115 com/sun/media/sound/SoftVoice.java

```
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.io.IOException;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

import javax.sound.midi.VoiceStatus;

/**
 * Software synthesizer voice class.
 *
 * @author Karl Helgason
 */
public class SoftVoice extends VoiceStatus {

    public int exclusiveClass = 0;
    public boolean releaseTriggered = false;
    private int noteOn_noteNumber = 0;
    private int noteOn_velocity = 0;
    private int noteOff_velocity = 0;
    private int delay = 0;
    protected ModelChannelMixer channelmixer = null;
    protected double tunedKey = 0;
    protected SoftTuning tuning = null;
    protected SoftChannel stealer_channel = null;
    protected ModelConnectionBlock[] stealer_extendedConnectionBlocks = null;
    protected SoftPerformer stealer_performer = null;
    protected ModelChannelMixer stealer_channelmixer = null;
    protected int stealer_voiceID = -1;
    protected int stealer_noteNumber = 0;
    protected int stealer_velocity = 0;
    protected boolean stealer_releaseTriggered = false;
    protected int voiceID = -1;
    protected boolean sustain = false;
    protected boolean sostenuto = false;
```

```
    protected boolean portamento = false;
    private SoftFilter filter_left;
    private SoftFilter filter_right;
    private SoftProcess eg = new SoftEnvelopeGenerator();
    private SoftProcess lfo = new SoftLowFrequencyOscillator();
    protected Map<String, SoftControl> objects =
            new HashMap<String, SoftControl>();
    protected SoftSynthesizer synthesizer;
    protected SoftInstrument instrument;
    protected SoftPerformer performer;
    protected SoftChannel softchannel = null;
    protected boolean on = false;
    private boolean audiostarted = false;
    private boolean started = false;
    private boolean stopping = false;
    private float osc_attenuation = 0.0f;
    private ModelOscillatorStream osc_stream;
    private int osc_stream_nrofchannels;
    private float[][] osc_buff = new float[2][];
    private boolean osc_stream_off_transmitted = false;
    private boolean out_mixer_end = false;
    private float out_mixer_left = 0;
    private float out_mixer_right = 0;
    private float out_mixer_effect1 = 0;
    private float out_mixer_effect2 = 0;
    private float last_out_mixer_left = 0;
    private float last_out_mixer_right = 0;
    private float last_out_mixer_effect1 = 0;
    private float last_out_mixer_effect2 = 0;
    protected ModelConnectionBlock[] extendedConnectionBlocks = null;
    private ModelConnectionBlock[] connections;
    // Last value added to destination
    private double[] connections_last = new double[50];
    // Pointer to source value
    private double[][][] connections_src = new double[50][3][];
    // Key-based override (if any)
    private int[][] connections_src_kc = new int[50][3];
    // Pointer to destination value
    private double[][] connections_dst = new double[50][];
    private boolean soundoff = false;
    private float lastMuteValue = 0;
    private float lastSoloMuteValue = 0;
    protected double[] co_noteon_keynumber = new double[1];
    protected double[] co_noteon_velocity = new double[1];
    protected double[] co_noteon_on = new double[1];
    private SoftControl co_noteon = new SoftControl() {
        double[] keynumber = co_noteon_keynumber;
        double[] velocity = co_noteon_velocity;
        double[] on = co_noteon_on;
        public double[] get(int instance, String name) {
            if (name == null)
                return null;
            if (name.equals("keynumber"))
                return keynumber;
            if (name.equals("velocity"))
                return velocity;
            if (name.equals("on"))
                return on;
            return null;
        }
    };
    private double[] co_mixer_active = new double[1];
```

```
123     private double[] co_mixer_gain = new double[1];
124     private double[] co_mixer_pan = new double[1];
125     private double[] co_mixer_balance = new double[1];
126     private double[] co_mixer_reverb = new double[1];
127     private double[] co_mixer_chorus = new double[1];
128     private SoftControl co_mixer = new SoftControl() {
129         double[] active = co_mixer_active;
130         double[] gain = co_mixer_gain;
131         double[] pan = co_mixer_pan;
132         double[] balance = co_mixer_balance;
133         double[] reverb = co_mixer_reverb;
134         double[] chorus = co_mixer_chorus;
135         public double[] get(int instance, String name) {
136             if (name == null)
137                 return null;
138             if (name.equals("active"))
139                 return active;
140             if (name.equals("gain"))
141                 return gain;
142             if (name.equals("pan"))
143                 return pan;
144             if (name.equals("balance"))
145                 return balance;
146             if (name.equals("reverb"))
147                 return reverb;
148             if (name.equals("chorus"))
149                 return chorus;
150             return null;
151         }
152     };
153     private double[] co_osc_pitch = new double[1];
154     private SoftControl co_osc = new SoftControl() {
155         double[] pitch = co_osc_pitch;
156         public double[] get(int instance, String name) {
157             if (name == null)
158                 return null;
159             if (name.equals("pitch"))
160                 return pitch;
161             return null;
162         }
163     };
164     private double[] co_filter_freq = new double[1];
165     private double[] co_filter_type = new double[1];
166     private double[] co_filter_q = new double[1];
167     private SoftControl co_filter = new SoftControl() {
168         double[] freq = co_filter_freq;
169         double[] ftype = co_filter_type;
170         double[] q = co_filter_q;
171         public double[] get(int instance, String name) {
172             if (name == null)
173                 return null;
174             if (name.equals("freq"))
175                 return freq;
176             if (name.equals("type"))
177                 return ftype;
178             if (name.equals("q"))
179                 return q;
180             return null;
181         }
182     };
183     protected SoftResamplerStreamer resampler;
184     private int nrofchannels;
```

```
185
186     public SoftVoice(SoftSynthesizer synth) {
187         synthesizer = synth;
188         filter_left = new SoftFilter(synth.getFormat().getSampleRate());
189         filter_right = new SoftFilter(synth.getFormat().getSampleRate());
190         nrofchannels = synth.getFormat().getChannels();
191     }
192
193     private int getValueKC(ModelIdentifier id) {
194         if (id.getObject().equals("midi_cc")) {
195             int ic = Integer.parseInt(id.getVariable());
196             if (ic != 0 && ic != 32) {
197                 if (ic < 120)
198                     return ic;
199             }
200         } else if (id.getObject().equals("midi_rpn")) {
201             if (id.getVariable().equals("1"))
202                 return 120; // Fine tuning
203             if (id.getVariable().equals("2"))
204                 return 121; // Coarse tuning
205         }
206         return -1;
207     }
208
209     private double[] getValue(ModelIdentifier id) {
210         SoftControl o = objects.get(id.getObject());
211         if (o == null)
212             return null;
213         return o.get(id.getInstance(), id.getVariable());
214     }
215
216     private double transformValue(double value, ModelSource src) {
217         if (src.getTransform() != null)
218             return src.getTransform().transform(value);
219         else
220             return value;
221     }
222
223     private double transformValue(double value, ModelDestination dst) {
224         if (dst.getTransform() != null)
225             return dst.getTransform().transform(value);
226         else
227             return value;
228     }
229
230     private double processKeyBasedController(double value, int keycontrol) {
231         if (keycontrol == -1)
232             return value;
233         if (softchannel.keybasedcontroller_active != null)
234             if (softchannel.keybasedcontroller_active[note] != null)
235                 if (softchannel.keybasedcontroller_active[note][keycontrol]) {
236                     double key_controlvalue =
237                             softchannel.keybasedcontroller_value[note][keycontrol];
238                     if (keycontrol == 10 || keycontrol == 91 || keycontrol == 93)
239                         return key_controlvalue;
240                     value += key_controlvalue * 2.0 - 1.0;
241                     if (value > 1)
242                         value = 1;
243                     else if (value < 0)
244                         value = 0;
245                 }
246         return value;
```

```
    }

    private void processConnection(int ix) {
        ModelConnectionBlock conn = connections[ix];
        double[][] src = connections_src[ix];
        double[] dst = connections_dst[ix];
        if (dst == null || Double.isInfinite(dst[0]))
            return;

        double value = conn.getScale();
        if (softchannel.keybasedcontroller_active == null) {
            ModelSource[] srcs = conn.getSources();
            for (int i = 0; i < srcs.length; i++) {
                value *= transformValue(src[i][0], srcs[i]);
                if (value == 0)
                    break;
            }
        } else {
            ModelSource[] srcs = conn.getSources();
            int[] src_kc = connections_src_kc[ix];
            for (int i = 0; i < srcs.length; i++) {
                value *= transformValue(processKeyBasedController(src[i][0],
                        src_kc[i]), srcs[i]);
                if (value == 0)
                    break;
            }
        }

        value = transformValue(value, conn.getDestination());
        dst[0] = dst[0] - connections_last[ix] + value;
        connections_last[ix] = value;
        // co_mixer_gain[0] = 0;
    }

    protected void updateTuning(SoftTuning newtuning) {
        tuning = newtuning;
        tunedKey = tuning.getTuning(note) / 100.0;
        if (!portamento) {
            co_noteon_keynumber[0] = tunedKey * (1.0 / 128.0);
            if(performer == null)
                return;
            int[] c = performer.midi_connections[4];
            if (c == null)
                return;
            for (int i = 0; i < c.length; i++)
                processConnection(c[i]);
        }
    }

    protected void setNote(int noteNumber) {
        note = noteNumber;
        tunedKey = tuning.getTuning(noteNumber) / 100.0;
    }

    protected void noteOn(int noteNumber, int velocity, int delay) {

        sustain = false;
        sostenuto = false;
        portamento = false;

        soundoff = false;
        on = true;
```

```
        active = true;
        started = true;
        // volume = velocity;

        noteOn_noteNumber = noteNumber;
        noteOn_velocity = velocity;
        this.delay = delay;

        lastMuteValue = 0;
        lastSoloMuteValue = 0;

        setNote(noteNumber);

        if (performer.forcedKeynumber)
            co_noteon_keynumber[0] = 0;
        else
            co_noteon_keynumber[0] = tunedKey * (1f / 128f);
        if (performer.forcedVelocity)
            co_noteon_velocity[0] = 0;
        else
            co_noteon_velocity[0] = velocity * (1f / 128f);
        co_mixer_active[0] = 0;
        co_mixer_gain[0] = 0;
        co_mixer_pan[0] = 0;
        co_mixer_balance[0] = 0;
        co_mixer_reverb[0] = 0;
        co_mixer_chorus[0] = 0;
        co_osc_pitch[0] = 0;
        co_filter_freq[0] = 0;
        co_filter_q[0] = 0;
        co_filter_type[0] = 0;
        co_noteon_on[0] = 1;

        eg.reset();
        lfo.reset();
        filter_left.reset();
        filter_right.reset();

        objects.put("master", synthesizer.getMainMixer().co_master);
        objects.put("eg", eg);
        objects.put("lfo", lfo);
        objects.put("noteon", co_noteon);
        objects.put("osc", co_osc);
        objects.put("mixer", co_mixer);
        objects.put("filter", co_filter);

        connections = performer.connections;

        if (connections_last == null
                || connections_last.length < connections.length) {
            connections_last = new double[connections.length];
        }
        if (connections_src == null
                || connections_src.length < connections.length) {
            connections_src = new double[connections.length][][];
            connections_src_kc = new int[connections.length][];
        }
        if (connections_dst == null
                || connections_dst.length < connections.length) {
            connections_dst = new double[connections.length][];
        }
        for (int i = 0; i < connections.length; i++) {
```

```
            ModelConnectionBlock conn = connections[i];
            connections_last[i] = 0;
            if (conn.getSources() != null) {
                ModelSource[] srcs = conn.getSources();
                if (connections_src[i] == null
                        || connections_src[i].length < srcs.length) {
                    connections_src[i] = new double[srcs.length][];
                    connections_src_kc[i] = new int[srcs.length];
                }
                double[][] src = connections_src[i];
                int[] src_kc = connections_src_kc[i];
                connections_src[i] = src;
                for (int j = 0; j < srcs.length; j++) {
                    src_kc[j] = getValueKC(srcs[j].getIdentifier());
                    src[j] = getValue(srcs[j].getIdentifier());
                }
            }

            if (conn.getDestination() != null)
                connections_dst[i] = getValue(conn.getDestination()
                        .getIdentifier());
            else
                connections_dst[i] = null;
        }

        for (int i = 0; i < connections.length; i++)
            processConnection(i);

        if (extendedConnectionBlocks != null) {
            for (ModelConnectionBlock connection: extendedConnectionBlocks) {
                double value = 0;

                if (softchannel.keybasedcontroller_active == null) {
                    for (ModelSource src: connection.getSources()) {
                        double x = getValue(src.getIdentifier())[0];
                        ModelTransform t = src.getTransform();
                        if (t == null)
                            value += x;
                        else
                            value += t.transform(x);
                    }
                } else {
                    for (ModelSource src: connection.getSources()) {
                        double x = getValue(src.getIdentifier())[0];
                        x = processKeyBasedController(x,
                                getValueKC(src.getIdentifier()));
                        ModelTransform t = src.getTransform();
                        if (t == null)
                            value += x;
                        else
                            value += t.transform(x);
                    }
                }

                ModelDestination dest = connection.getDestination();
                ModelTransform t = dest.getTransform();
                if (t != null)
                    value = t.transform(value);
                getValue(dest.getIdentifier())[0] += value;
            }
        }

```

```
        eg.init(synthesizer);
        lfo.init(synthesizer);

    }

    protected void setPolyPressure(int pressure) {
        if(performer == null)
            return;
        int[] c = performer.midi_connections[2];
        if (c == null)
            return;
        for (int i = 0; i < c.length; i++)
            processConnection(c[i]);
    }

    protected void setChannelPressure(int pressure) {
        if(performer == null)
            return;
        int[] c = performer.midi_connections[1];
        if (c == null)
            return;
        for (int i = 0; i < c.length; i++)
            processConnection(c[i]);
    }

    protected void controlChange(int controller, int value) {
        if(performer == null)
            return;
        int[] c = performer.midi_ctrl_connections[controller];
        if (c == null)
            return;
        for (int i = 0; i < c.length; i++)
            processConnection(c[i]);
    }

    protected void nrpnChange(int controller, int value) {
        if(performer == null)
            return;
        int[] c = performer.midi_nrpn_connections.get(controller);
        if (c == null)
            return;
        for (int i = 0; i < c.length; i++)
            processConnection(c[i]);
    }

    protected void rpnChange(int controller, int value) {
        if(performer == null)
            return;
        int[] c = performer.midi_rpn_connections.get(controller);
        if (c == null)
            return;
        for (int i = 0; i < c.length; i++)
            processConnection(c[i]);
    }

    protected void setPitchBend(int bend) {
        if(performer == null)
            return;
        int[] c = performer.midi_connections[0];
        if (c == null)
            return;
        for (int i = 0; i < c.length; i++)
```

```
            processConnection(c[i]);
    }

    protected void setMute(boolean mute) {
        co_mixer_gain[0] -= lastMuteValue;
        lastMuteValue = mute ? -960 : 0;
        co_mixer_gain[0] += lastMuteValue;
    }

    protected void setSoloMute(boolean mute) {
        co_mixer_gain[0] -= lastSoloMuteValue;
        lastSoloMuteValue = mute ? -960 : 0;
        co_mixer_gain[0] += lastSoloMuteValue;
    }

    protected void shutdown() {
        if (co_noteon_on[0] < -0.5)
            return;
        on = false;

        co_noteon_on[0] = -1;

        if(performer == null)
            return;
        int[] c = performer.midi_connections[3];
        if (c == null)
            return;
        for (int i = 0; i < c.length; i++)
            processConnection(c[i]);
    }

    protected void soundOff() {
        on = false;
        soundoff = true;
    }

    protected void noteOff(int velocity) {
        if (!on)
            return;
        on = false;

        noteOff_velocity = velocity;

        if (softchannel.sustain) {
            sustain = true;
            return;
        }
        if (sostenuto)
            return;

        co_noteon_on[0] = 0;

        if(performer == null)
            return;
        int[] c = performer.midi_connections[3];
        if (c == null)
            return;
        for (int i = 0; i < c.length; i++)
            processConnection(c[i]);
    }

    protected void redamp() {
```

```
        if (co_noteon_on[0] > 0.5)
            return;
        if (co_noteon_on[0] < -0.5)
            return; // don't redamp notes in shutdown stage

        sustain = true;
        co_noteon_on[0] = 1;

        if(performer == null)
            return;
        int[] c = performer.midi_connections[3];
        if (c == null)
            return;
        for (int i = 0; i < c.length; i++)
            processConnection(c[i]);
    }

    protected void processControlLogic() {
        if (stopping) {
            active = false;
            stopping = false;
            audiostarted = false;
            instrument = null;
            performer = null;
            connections = null;
            extendedConnectionBlocks = null;
            channelmixer = null;
            if (osc_stream != null)
                try {
                    osc_stream.close();
                } catch (IOException e) {
                    //e.printStackTrace();
                }

            if (stealer_channel != null) {
                stealer_channel.initVoice(this, stealer_performer,
                        stealer_voiceID, stealer_noteNumber, stealer_velocity, 0,
                        stealer_extendedConnectionBlocks, stealer_channelmixer,
                        stealer_releaseTriggered);
                stealer_releaseTriggered = false;
                stealer_channel = null;
                stealer_performer = null;
                stealer_voiceID = -1;
                stealer_noteNumber = 0;
                stealer_velocity = 0;
                stealer_extendedConnectionBlocks = null;
                stealer_channelmixer = null;
            }
        }
        if (started) {
            audiostarted = true;

            ModelOscillator osc = performer.oscillators[0];

            osc_stream_off_transmitted = false;
            if (osc instanceof ModelWavetable) {
                try {
                    resampler.open((ModelWavetable)osc,
                            synthesizer.getFormat().getSampleRate());
                    osc_stream = resampler;
                } catch (IOException e) {
                    //e.printStackTrace();
```

```
619                 }
620             } else {
621                 osc_stream = osc.open(synthesizer.getFormat().getSampleRate());
622             }
623             osc_attenuation = osc.getAttenuation();
624             osc_stream_nrofchannels = osc.getChannels();
625             if (osc_buff == null || osc_buff.length < osc_stream_nrofchannels)
626                 osc_buff = new float[osc_stream_nrofchannels][];
627 
628             if (osc_stream != null)
629                 osc_stream.noteOn(softchannel, this, noteOn_noteNumber,
630                         noteOn_velocity);
631 
632 
633         }
634         if (audiostarted) {
635             if (portamento) {
636                 double note_delta = tunedKey - (co_noteon_keynumber[0] * 128);
637                 double note_delta_a = Math.abs(note_delta);
638                 if (note_delta_a < 0.0000000001) {
639                     co_noteon_keynumber[0] = tunedKey * (1.0 / 128.0);
640                     portamento = false;
641                 } else {
642                     if (note_delta_a > softchannel.portamento_time)
643                         note_delta = Math.signum(note_delta)
644                                 * softchannel.portamento_time;
645                     co_noteon_keynumber[0] += note_delta * (1.0 / 128.0);
646                 }
647 
648                 int[] c = performer.midi_connections[4];
649                 if (c == null)
650                     return;
651                 for (int i = 0; i < c.length; i++)
652                     processConnection(c[i]);
653             }
654 
655             eg.processControlLogic();
656             lfo.processControlLogic();
657 
658             for (int i = 0; i < performer.ctrl_connections.length; i++)
659                 processConnection(performer.ctrl_connections[i]);
660 
661             osc_stream.setPitch((float)co_osc_pitch[0]);
662 
663             int filter_type = (int)co_filter_type[0];
664             double filter_freq;
665 
666             if (co_filter_freq[0] == 13500.0)
667                 filter_freq = 19912.126958213175;
668             else
669                 filter_freq = 440.0 * Math.exp(
670                         ((co_filter_freq[0]) - 6900.0) *
671                         (Math.log(2.0) / 1200.0));
672             /*
673             filter_freq = 440.0 * Math.pow(2.0,
674             ((co_filter_freq[0]) - 6900.0) / 1200.0);*/
675             /*
676              * double velocity = co_noteon_velocity[0]; if(velocity < 0.5)
677              * filter_freq *= ((velocity * 2)*0.75 + 0.25);
678              */
679 
680             double q = co_filter_q[0] / 10.0;
```

```
681             filter_left.setFilterType(filter_type);
682             filter_left.setFrequency(filter_freq);
683             filter_left.setResonance(q);
684             filter_right.setFilterType(filter_type);
685             filter_right.setFrequency(filter_freq);
686             filter_right.setResonance(q);
687             /*
688             float gain = (float) Math.pow(10,
689             (-osc_attenuation + co_mixer_gain[0]) / 200.0);
690              */
691             float gain = (float)Math.exp(
692                     (-osc_attenuation + co_mixer_gain[0])*(Math.log(10) / 200.0));
693
694             if (co_mixer_gain[0] <= -960)
695                 gain = 0;
696
697             if (soundoff) {
698                 stopping = true;
699                 gain = 0;
700                 /*
701                  * if(co_mixer_gain[0] > -960)
702                  *   co_mixer_gain[0] -= 960;
703                  */
704             }
705
706             volume = (int)(Math.sqrt(gain) * 128);
707
708             // gain *= 0.2;
709
710             double pan = co_mixer_pan[0] * (1.0 / 1000.0);
711             // System.out.println("pan = " + pan);
712             if (pan < 0)
713                 pan = 0;
714             else if (pan > 1)
715                 pan = 1;
716
717             if (pan == 0.5) {
718                 out_mixer_left = gain * 0.70710678118654476f;
719                 out_mixer_right = out_mixer_left;
720             } else {
721                 out_mixer_left = gain * (float)Math.cos(pan * Math.PI * 0.5);
722                 out_mixer_right = gain * (float)Math.sin(pan * Math.PI * 0.5);
723             }
724
725             double balance = co_mixer_balance[0] * (1.0 / 1000.0);
726             if (balance != 0.5) {
727                 if (balance > 0.5)
728                     out_mixer_left *= (1 - balance) * 2;
729                 else
730                     out_mixer_right *= balance * 2;
731             }
732
733             if (synthesizer.reverb_on) {
734                 out_mixer_effect1 = (float)(co_mixer_reverb[0] * (1.0 / 1000.0));
735                 out_mixer_effect1 *= gain;
736             } else
737                 out_mixer_effect1 = 0;
738             if (synthesizer.chorus_on) {
739                 out_mixer_effect2 = (float)(co_mixer_chorus[0] * (1.0 / 1000.0));
740                 out_mixer_effect2 *= gain;
741             } else
742                 out_mixer_effect2 = 0;
```

```
            out_mixer_end = co_mixer_active[0] < 0.5;

            if (!on)
                if (!osc_stream_off_transmitted) {
                    osc_stream_off_transmitted = true;
                    if (osc_stream != null)
                        osc_stream.noteOff(noteOff_velocity);
                }

        }
        if (started) {
            last_out_mixer_left = out_mixer_left;
            last_out_mixer_right = out_mixer_right;
            last_out_mixer_effect1 = out_mixer_effect1;
            last_out_mixer_effect2 = out_mixer_effect2;
            started = false;
        }

    }

    protected void mixAudioStream(SoftAudioBuffer in, SoftAudioBuffer out,
            SoftAudioBuffer dout,
            float amp_from, float amp_to) {
        int bufferlen = in.getSize();
        if (amp_from < 0.000000001 && amp_to < 0.000000001)
            return;
        if(dout != null && delay != 0)
        {
            if (amp_from == amp_to) {
                float[] fout = out.array();
                float[] fin = in.array();
                int j = 0;
                for (int i = delay; i < bufferlen; i++)
                    fout[i] += fin[j++] * amp_to;
                fout = dout.array();
                for (int i = 0; i < delay; i++)
                    fout[i] += fin[j++] * amp_to;
            } else {
                float amp = amp_from;
                float amp_delta = (amp_to - amp_from) / bufferlen;
                float[] fout = out.array();
                float[] fin = in.array();
                int j = 0;
                for (int i = delay; i < bufferlen; i++) {
                    amp += amp_delta;
                    fout[i] += fin[j++] * amp;
                }
                fout = dout.array();
                for (int i = 0; i < delay; i++) {
                    amp += amp_delta;
                    fout[i] += fin[j++] * amp;
                }
            }
        }
        else
        {
            if (amp_from == amp_to) {
                float[] fout = out.array();
                float[] fin = in.array();
                for (int i = 0; i < bufferlen; i++)
                    fout[i] += fin[i] * amp_to;
            } else {
```

```
                float amp = amp_from;
                float amp_delta = (amp_to - amp_from) / bufferlen;
                float[] fout = out.array();
                float[] fin = in.array();
                for (int i = 0; i < bufferlen; i++) {
                    amp += amp_delta;
                    fout[i] += fin[i] * amp;
                }
            }
        }

    }

    protected void processAudioLogic(SoftAudioBuffer[] buffer) {
        if (!audiostarted)
            return;

        int bufferlen = buffer[0].getSize();

        try {
            osc_buff[0] = buffer[SoftMainMixer.CHANNEL_LEFT_DRY].array();
            if (nrofchannels != 1)
                osc_buff[1] = buffer[SoftMainMixer.CHANNEL_RIGHT_DRY].array();
            int ret = osc_stream.read(osc_buff, 0, bufferlen);
            if (ret == -1) {
                stopping = true;
                return;
            }
            if (ret != bufferlen) {
                Arrays.fill(osc_buff[0], ret, bufferlen, 0f);
                if (nrofchannels != 1)
                    Arrays.fill(osc_buff[1], ret, bufferlen, 0f);
            }

        } catch (IOException e) {
            //e.printStackTrace();
        }

        SoftAudioBuffer left = buffer[SoftMainMixer.CHANNEL_LEFT];
        SoftAudioBuffer right = buffer[SoftMainMixer.CHANNEL_RIGHT];
        SoftAudioBuffer mono = buffer[SoftMainMixer.CHANNEL_MONO];
        SoftAudioBuffer eff1 = buffer[SoftMainMixer.CHANNEL_EFFECT1];
        SoftAudioBuffer eff2 = buffer[SoftMainMixer.CHANNEL_EFFECT2];

        SoftAudioBuffer dleft = buffer[SoftMainMixer.CHANNEL_DELAY_LEFT];
        SoftAudioBuffer dright = buffer[SoftMainMixer.CHANNEL_DELAY_RIGHT];
        SoftAudioBuffer dmono = buffer[SoftMainMixer.CHANNEL_DELAY_MONO];
        SoftAudioBuffer deff1 = buffer[SoftMainMixer.CHANNEL_DELAY_EFFECT1];
        SoftAudioBuffer deff2 = buffer[SoftMainMixer.CHANNEL_DELAY_EFFECT2];

        SoftAudioBuffer leftdry = buffer[SoftMainMixer.CHANNEL_LEFT_DRY];
        SoftAudioBuffer rightdry = buffer[SoftMainMixer.CHANNEL_RIGHT_DRY];

        if (osc_stream_nrofchannels == 1)
            rightdry = null;

        if (!Double.isInfinite(co_filter_freq[0])) {
            filter_left.processAudio(leftdry);
            if (rightdry != null)
                filter_right.processAudio(rightdry);
        }

```

```
867         if (nrofchannels == 1) {
868             out_mixer_left = (out_mixer_left + out_mixer_right) / 2;
869             mixAudioStream(leftdry, left, dleft, last_out_mixer_left, out_mixer_left);
870             if (rightdry != null)
871                 mixAudioStream(rightdry, left, dleft, last_out_mixer_left,
872                         out_mixer_left);
873         } else {
874             if(rightdry == null &&
875                     last_out_mixer_left == last_out_mixer_right &&
876                     out_mixer_left == out_mixer_right)
877             {
878                 mixAudioStream(leftdry, mono, dmono, last_out_mixer_left, out_mixer_left);
879             }
880             else
881             {
882                 mixAudioStream(leftdry, left, dleft, last_out_mixer_left, out_mixer_left);
883                 if (rightdry != null)
884                     mixAudioStream(rightdry, right, dright, last_out_mixer_right,
885                         out_mixer_right);
886                 else
887                     mixAudioStream(leftdry, right, dright, last_out_mixer_right,
888                         out_mixer_right);
889             }
890         }
891 
892         if (rightdry == null) {
893             mixAudioStream(leftdry, eff1, deff1, last_out_mixer_effect1,
894                     out_mixer_effect1);
895             mixAudioStream(leftdry, eff2, deff2, last_out_mixer_effect2,
896                     out_mixer_effect2);
897         } else {
898             mixAudioStream(leftdry, eff1, deff1, last_out_mixer_effect1 * 0.5f,
899                     out_mixer_effect1 * 0.5f);
900             mixAudioStream(leftdry, eff2, deff2, last_out_mixer_effect2 * 0.5f,
901                     out_mixer_effect2 * 0.5f);
902             mixAudioStream(rightdry, eff1, deff1, last_out_mixer_effect1 * 0.5f,
903                     out_mixer_effect1 * 0.5f);
904             mixAudioStream(rightdry, eff2, deff2, last_out_mixer_effect2 * 0.5f,
905                     out_mixer_effect2 * 0.5f);
906         }
907 
908         last_out_mixer_left = out_mixer_left;
909         last_out_mixer_right = out_mixer_right;
910         last_out_mixer_effect1 = out_mixer_effect1;
911         last_out_mixer_effect2 = out_mixer_effect2;
912 
913         if (out_mixer_end) {
914             stopping = true;
915         }
916 
917     }
918 }
```

116 com/sun/media/sound/WaveExtensibleFileReader.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.util.HashMap;
import java.util.Map;

import javax.sound.sampled.AudioFileFormat;
import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.UnsupportedAudioFileException;
import javax.sound.sampled.AudioFormat.Encoding;
import javax.sound.sampled.spi.AudioFileReader;

/**
 * WAVE file reader for files using format WAVE_FORMAT_EXTENSIBLE (0xFFFE).
 *
 * @author Karl Helgason
 */
public class WaveExtensibleFileReader extends AudioFileReader {

    static private class GUID {
        long i1;

        int s1;

        int s2;

        int x1;

        int x2;
```

```
61
62          int x3;
63
64          int x4;
65
66          int x5;
67
68          int x6;
69
70          int x7;
71
72          int x8;
73
74          private GUID() {
75          }
76
77          public GUID(long i1, int s1, int s2, int x1, int x2, int x3, int x4,
78                  int x5, int x6, int x7, int x8) {
79              this.i1 = i1;
80              this.s1 = s1;
81              this.s2 = s2;
82              this.x1 = x1;
83              this.x2 = x2;
84              this.x3 = x3;
85              this.x4 = x4;
86              this.x5 = x5;
87              this.x6 = x6;
88              this.x7 = x7;
89              this.x8 = x8;
90          }
91
92          public static GUID read(RIFFReader riff) throws IOException {
93              GUID d = new GUID();
94              d.i1 = riff.readUnsignedInt();
95              d.s1 = riff.readUnsignedShort();
96              d.s2 = riff.readUnsignedShort();
97              d.x1 = riff.readUnsignedByte();
98              d.x2 = riff.readUnsignedByte();
99              d.x3 = riff.readUnsignedByte();
100             d.x4 = riff.readUnsignedByte();
101             d.x5 = riff.readUnsignedByte();
102             d.x6 = riff.readUnsignedByte();
103             d.x7 = riff.readUnsignedByte();
104             d.x8 = riff.readUnsignedByte();
105             return d;
106         }
107
108         public int hashCode() {
109             return (int) i1;
110         }
111
112         public boolean equals(Object obj) {
113             if (!(obj instanceof GUID))
114                 return false;
115             GUID t = (GUID) obj;
116             if (i1 != t.i1)
117                 return false;
118             if (s1 != t.s1)
119                 return false;
120             if (s2 != t.s2)
121                 return false;
122             if (x1 != t.x1)
```

```
            return false;
        if (x2 != t.x2)
            return false;
        if (x3 != t.x3)
            return false;
        if (x4 != t.x4)
            return false;
        if (x5 != t.x5)
            return false;
        if (x6 != t.x6)
            return false;
        if (x7 != t.x7)
            return false;
        if (x8 != t.x8)
            return false;
        return true;
    }

}

private static String[] channelnames = { "FL", "FR", "FC", "LF",
        "BL",
        "BR", // 5.1
        "FLC", "FLR", "BC", "SL", "SR", "TC", "TFL", "TFC", "TFR", "TBL",
        "TBC", "TBR" };

private static String[] allchannelnames = { "w1", "w2", "w3", "w4", "w5",
        "w6", "w7", "w8", "w9", "w10", "w11", "w12", "w13", "w14", "w15",
        "w16", "w17", "w18", "w19", "w20", "w21", "w22", "w23", "w24",
        "w25", "w26", "w27", "w28", "w29", "w30", "w31", "w32", "w33",
        "w34", "w35", "w36", "w37", "w38", "w39", "w40", "w41", "w42",
        "w43", "w44", "w45", "w46", "w47", "w48", "w49", "w50", "w51",
        "w52", "w53", "w54", "w55", "w56", "w57", "w58", "w59", "w60",
        "w61", "w62", "w63", "w64" };

private static GUID SUBTYPE_PCM = new GUID(0x00000001, 0x0000, 0x0010,
        0x80, 0x00, 0x00, 0xaa, 0x00, 0x38, 0x9b, 0x71);

private static GUID SUBTYPE_IEEE_FLOAT = new GUID(0x00000003, 0x0000,
        0x0010, 0x80, 0x00, 0x00, 0xaa, 0x00, 0x38, 0x9b, 0x71);

private String decodeChannelMask(long channelmask) {
    StringBuffer sb = new StringBuffer();
    long m = 1;
    for (int i = 0; i < allchannelnames.length; i++) {
        if ((channelmask & m) != 0L) {
            if (i < channelnames.length) {
                sb.append(channelnames[i] + "␣");
            } else {
                sb.append(allchannelnames[i] + "␣");
            }
        }
        m *= 2L;
    }
    if (sb.length() == 0)
        return null;
    return sb.substring(0, sb.length() - 1);

}

public AudioFileFormat getAudioFileFormat(InputStream stream)
        throws UnsupportedAudioFileException, IOException {
```

```
185
186         stream.mark(200);
187         AudioFileFormat format;
188         try {
189             format = internal_getAudioFileFormat(stream);
190         } finally {
191             stream.reset();
192         }
193         return format;
194     }
195
196     private AudioFileFormat internal_getAudioFileFormat(InputStream stream)
197             throws UnsupportedAudioFileException, IOException {
198
199         RIFFReader riffiterator = new RIFFReader(stream);
200         if (!riffiterator.getFormat().equals("RIFF"))
201             throw new UnsupportedAudioFileException();
202         if (!riffiterator.getType().equals("WAVE"))
203             throw new UnsupportedAudioFileException();
204
205         boolean fmt_found = false;
206         boolean data_found = false;
207
208         int channels = 1;
209         long samplerate = 1;
210         // long framerate = 1;
211         int framesize = 1;
212         int bits = 1;
213         int validBitsPerSample = 1;
214         long channelMask = 0;
215         GUID subFormat = null;
216
217         while (riffiterator.hasNextChunk()) {
218             RIFFReader chunk = riffiterator.nextChunk();
219
220             if (chunk.getFormat().equals("fmt ")) {
221                 fmt_found = true;
222
223                 int format = chunk.readUnsignedShort();
224                 if (format != 0xFFFE)
225                     throw new UnsupportedAudioFileException(); // WAVE_FORMAT_EXTENSIBLE
226                 // only
227                 channels = chunk.readUnsignedShort();
228                 samplerate = chunk.readUnsignedInt();
229                 /* framerate = */chunk.readUnsignedInt();
230                 framesize = chunk.readUnsignedShort();
231                 bits = chunk.readUnsignedShort();
232                 int cbSize = chunk.readUnsignedShort();
233                 if (cbSize != 22)
234                     throw new UnsupportedAudioFileException();
235                 validBitsPerSample = chunk.readUnsignedShort();
236                 if (validBitsPerSample > bits)
237                     throw new UnsupportedAudioFileException();
238                 channelMask = chunk.readUnsignedInt();
239                 subFormat = GUID.read(chunk);
240
241             }
242             if (chunk.getFormat().equals("data")) {
243                 data_found = true;
244                 break;
245             }
246         }
```

```
247
248         if (!fmt_found)
249             throw new UnsupportedAudioFileException();
250         if (!data_found)
251             throw new UnsupportedAudioFileException();
252
253         Map<String, Object> p = new HashMap<String, Object>();
254         String s_channelmask = decodeChannelMask(channelMask);
255         if (s_channelmask != null)
256             p.put("channelOrder", s_channelmask);
257         if (channelMask != 0)
258             p.put("channelMask", channelMask);
259         // validBitsPerSample is only informational for PCM data,
260         // data is still encode according to SampleSizeInBits.
261         p.put("validBitsPerSample", validBitsPerSample);
262
263         AudioFormat audioformat = null;
264         if (subFormat.equals(SUBTYPE_PCM)) {
265             if (bits == 8) {
266                 audioformat = new AudioFormat(Encoding.PCM_UNSIGNED,
267                         samplerate, bits, channels, framesize, samplerate,
268                         false, p);
269             } else {
270                 audioformat = new AudioFormat(Encoding.PCM_SIGNED, samplerate,
271                         bits, channels, framesize, samplerate, false, p);
272             }
273         } else if (subFormat.equals(SUBTYPE_IEEE_FLOAT)) {
274             audioformat = new AudioFormat(AudioFloatConverter.PCM_FLOAT,
275                     samplerate, bits, channels, framesize, samplerate, false, p);
276         } else
277             throw new UnsupportedAudioFileException();
278
279         AudioFileFormat fileformat = new AudioFileFormat(
280                 AudioFileFormat.Type.WAVE, audioformat,
281                 AudioSystem.NOT_SPECIFIED);
282         return fileformat;
283     }
284
285     public AudioInputStream getAudioInputStream(InputStream stream)
286             throws UnsupportedAudioFileException, IOException {
287
288         AudioFileFormat format = getAudioFileFormat(stream);
289         RIFFReader riffiterator = new RIFFReader(stream);
290         if (!riffiterator.getFormat().equals("RIFF"))
291             throw new UnsupportedAudioFileException();
292         if (!riffiterator.getType().equals("WAVE"))
293             throw new UnsupportedAudioFileException();
294         while (riffiterator.hasNextChunk()) {
295             RIFFReader chunk = riffiterator.nextChunk();
296             if (chunk.getFormat().equals("data")) {
297                 return new AudioInputStream(chunk, format.getFormat(), chunk
298                         .getSize());
299             }
300         }
301         throw new UnsupportedAudioFileException();
302     }
303
304     public AudioFileFormat getAudioFileFormat(URL url)
305             throws UnsupportedAudioFileException, IOException {
306         InputStream stream = url.openStream();
307         AudioFileFormat format;
308         try {
```

```
            format = getAudioFileFormat(new BufferedInputStream(stream));
        } finally {
            stream.close();
        }
        return format;
    }

    public AudioFileFormat getAudioFileFormat(File file)
            throws UnsupportedAudioFileException, IOException {
        InputStream stream = new FileInputStream(file);
        AudioFileFormat format;
        try {
            format = getAudioFileFormat(new BufferedInputStream(stream));
        } finally {
            stream.close();
        }
        return format;
    }

    public AudioInputStream getAudioInputStream(URL url)
            throws UnsupportedAudioFileException, IOException {
        return getAudioInputStream(new BufferedInputStream(url.openStream()));
    }

    public AudioInputStream getAudioInputStream(File file)
            throws UnsupportedAudioFileException, IOException {
        return getAudioInputStream(new BufferedInputStream(new FileInputStream(
                file)));
    }

}
```

117 com/sun/media/sound/WaveFloatFileReader.java

```java
/*
 * Copyright 2007 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;

import javax.sound.sampled.AudioFileFormat;
import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.UnsupportedAudioFileException;
import javax.sound.sampled.spi.AudioFileReader;

/**
 * Floating-point encoded (format 3) WAVE file loader.
 *
 * @author Karl Helgason
 */
public class WaveFloatFileReader extends AudioFileReader {

    public AudioFileFormat getAudioFileFormat(InputStream stream)
            throws UnsupportedAudioFileException, IOException {

        stream.mark(200);
        AudioFileFormat format;
        try {
            format = internal_getAudioFileFormat(stream);
        } finally {
            stream.reset();
        }
        return format;
    }

```

```
    private AudioFileFormat internal_getAudioFileFormat(InputStream stream)
            throws UnsupportedAudioFileException, IOException {

        RIFFReader riffiterator = new RIFFReader(stream);
        if (!riffiterator.getFormat().equals("RIFF"))
            throw new UnsupportedAudioFileException();
        if (!riffiterator.getType().equals("WAVE"))
            throw new UnsupportedAudioFileException();

        boolean fmt_found = false;
        boolean data_found = false;

        int channels = 1;
        long samplerate = 1;
        int framesize = 1;
        int bits = 1;

        while (riffiterator.hasNextChunk()) {
            RIFFReader chunk = riffiterator.nextChunk();

            if (chunk.getFormat().equals("fmt ")) {
                fmt_found = true;

                int format = chunk.readUnsignedShort();
                if (format != 3) // WAVE_FORMAT_IEEE_FLOAT only
                    throw new UnsupportedAudioFileException();
                channels = chunk.readUnsignedShort();
                samplerate = chunk.readUnsignedInt();
                /* framerate = */chunk.readUnsignedInt();
                framesize = chunk.readUnsignedShort();
                bits = chunk.readUnsignedShort();
            }
            if (chunk.getFormat().equals("data")) {
                data_found = true;
                break;
            }
        }

        if (!fmt_found)
            throw new UnsupportedAudioFileException();
        if (!data_found)
            throw new UnsupportedAudioFileException();

        AudioFormat audioformat = new AudioFormat(
                AudioFloatConverter.PCM_FLOAT, samplerate, bits, channels,
                framesize, samplerate, false);
        AudioFileFormat fileformat = new AudioFileFormat(
                AudioFileFormat.Type.WAVE, audioformat,
                AudioSystem.NOT_SPECIFIED);
        return fileformat;
    }

    public AudioInputStream getAudioInputStream(InputStream stream)
            throws UnsupportedAudioFileException, IOException {

        AudioFileFormat format = getAudioFileFormat(stream);
        RIFFReader riffiterator = new RIFFReader(stream);
        if (!riffiterator.getFormat().equals("RIFF"))
            throw new UnsupportedAudioFileException();
        if (!riffiterator.getType().equals("WAVE"))
            throw new UnsupportedAudioFileException();
        while (riffiterator.hasNextChunk()) {
```

```
            RIFFReader chunk = riffiterator.nextChunk();
            if (chunk.getFormat().equals("data")) {
                return new AudioInputStream(chunk, format.getFormat(),
                        chunk.getSize());
            }
        }
        throw new UnsupportedAudioFileException();
    }

    public AudioFileFormat getAudioFileFormat(URL url)
            throws UnsupportedAudioFileException, IOException {
        InputStream stream = url.openStream();
        AudioFileFormat format;
        try {
            format = getAudioFileFormat(new BufferedInputStream(stream));
        } finally {
            stream.close();
        }
        return format;
    }

    public AudioFileFormat getAudioFileFormat(File file)
            throws UnsupportedAudioFileException, IOException {
        InputStream stream = new FileInputStream(file);
        AudioFileFormat format;
        try {
            format = getAudioFileFormat(new BufferedInputStream(stream));
        } finally {
            stream.close();
        }
        return format;
    }

    public AudioInputStream getAudioInputStream(URL url)
            throws UnsupportedAudioFileException, IOException {
        return getAudioInputStream(new BufferedInputStream(url.openStream()));
    }

    public AudioInputStream getAudioInputStream(File file)
            throws UnsupportedAudioFileException, IOException {
        return getAudioInputStream(new BufferedInputStream(new FileInputStream(
                file)));
    }
}
```

118 com/sun/media/sound/WaveFloatFileWriter.java

```java
/*
 * Copyright 2008 Sun Microsystems, Inc.  All Rights Reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Sun designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Sun in the LICENSE file that accompanied this code.
 *
 * This code 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 General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
 * CA 95054 USA or visit www.sun.com if you need additional information or
 * have any questions.
 */
package com.sun.media.sound;

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

import javax.sound.sampled.AudioFileFormat;
import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.AudioFileFormat.Type;
import javax.sound.sampled.spi.AudioFileWriter;

/**
 * Floating-point encoded (format 3) WAVE file writer.
 *
 * @author Karl Helgason
 */
public class WaveFloatFileWriter extends AudioFileWriter {

    public Type[] getAudioFileTypes() {
        return new Type[] { Type.WAVE };
    }

    public Type[] getAudioFileTypes(AudioInputStream stream) {

        if (!stream.getFormat().getEncoding().equals(
                AudioFloatConverter.PCM_FLOAT))
            return new Type[0];
        return new Type[] { Type.WAVE };
    }

    private void checkFormat(AudioFileFormat.Type type, AudioInputStream stream) {
        if (!Type.WAVE.equals(type))
            throw new IllegalArgumentException("File type " + type
                    + " not supported.");
```

```
        if (!stream.getFormat().getEncoding().equals(
                AudioFloatConverter.PCM_FLOAT))
            throw new IllegalArgumentException("File␣format␣"
                    + stream.getFormat() + "␣not␣supported.");
    }

    public void write(AudioInputStream stream, RIFFWriter writer)
            throws IOException {

        RIFFWriter fmt_chunk = writer.writeChunk("fmt␣");

        AudioFormat format = stream.getFormat();
        fmt_chunk.writeUnsignedShort(3); // WAVE_FORMAT_IEEE_FLOAT
        fmt_chunk.writeUnsignedShort(format.getChannels());
        fmt_chunk.writeUnsignedInt((int) format.getSampleRate());
        fmt_chunk.writeUnsignedInt(((int) format.getFrameRate())
                * format.getFrameSize());
        fmt_chunk.writeUnsignedShort(format.getFrameSize());
        fmt_chunk.writeUnsignedShort(format.getSampleSizeInBits());
        fmt_chunk.close();
        RIFFWriter data_chunk = writer.writeChunk("data");
        byte[] buff = new byte[1024];
        int len;
        while ((len = stream.read(buff, 0, buff.length)) != -1)
            data_chunk.write(buff, 0, len);
        data_chunk.close();
    }

    private static class NoCloseOutputStream extends OutputStream {
        OutputStream out;

        public NoCloseOutputStream(OutputStream out) {
            this.out = out;
        }

        public void write(int b) throws IOException {
            out.write(b);
        }

        public void flush() throws IOException {
            out.flush();
        }

        public void write(byte[] b, int off, int len) throws IOException {
            out.write(b, off, len);
        }

        public void write(byte[] b) throws IOException {
            out.write(b);
        }
    }

    private AudioInputStream toLittleEndian(AudioInputStream ais) {
        AudioFormat format = ais.getFormat();
        AudioFormat targetFormat = new AudioFormat(format.getEncoding(), format
                .getSampleRate(), format.getSampleSizeInBits(), format
                .getChannels(), format.getFrameSize(), format.getFrameRate(),
                false);
        return AudioSystem.getAudioInputStream(targetFormat, ais);
    }

    public int write(AudioInputStream stream, Type fileType, OutputStream out)
```

```
            throws IOException {

        checkFormat(fileType, stream);
        if (stream.getFormat().isBigEndian())
            stream = toLittleEndian(stream);
        RIFFWriter writer = new RIFFWriter(new NoCloseOutputStream(out), "WAVE");
        write(stream, writer);
        int fpointer = (int) writer.getFilePointer();
        writer.close();
        return fpointer;
    }

    public int write(AudioInputStream stream, Type fileType, File out)
            throws IOException {
        checkFormat(fileType, stream);
        if (stream.getFormat().isBigEndian())
            stream = toLittleEndian(stream);
        RIFFWriter writer = new RIFFWriter(out, "WAVE");
        write(stream, writer);
        int fpointer = (int) writer.getFilePointer();
        writer.close();
        return fpointer;
    }

}
```

119 simplemidiplayer/ConfigDialog.java

```
/*
 * Copyright (c) 2007 by Karl Helgason
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * - Redistributions of source code must retain the above copyright notice,
 *   this list of conditions and the following disclaimer.
 * - Redistributions in binary form must reproduce the above copyright
 *   notice, this list of conditions and the following disclaimer in the
 *   documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package simplemidiplayer;

import java.awt.BorderLayout;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.util.ArrayList;
import java.util.Properties;

import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.Line;
import javax.sound.sampled.Mixer;
import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JComponent;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.KeyStroke;

public class ConfigDialog extends JDialog {

  private static final long serialVersionUID = 1L;

  boolean isok = false;

  private String getValueFromList(JComboBox box) {
```

```
    if (box.isEditable())
      return box.getEditor().getItem().toString();
    return box.getSelectedItem().toString();
  }

  private void selectValueInList(JComboBox box, String value) {
    if (value == null)
      return;
    if (box.isEditable()) {
      box.getEditor().setItem(value);
    } else {
      for (int i = 0; i < box.getItemCount(); i++) {
        if (box.getItemAt(i).equals(value)) {
          box.setSelectedIndex(i);
          return;
        }
      }
    }
  }

  public boolean isOK() {
    return isok;
  }

  public ConfigDialog(JFrame parent) {
    super(parent);
    setSize(550, 400);
    // setLocationByPlatform(true);
    setModal(true);
    setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
    setTitle("Gervill - MIDI Player - Config");

    ArrayList<String> dev_list = new ArrayList<String>();
    dev_list.add("(default)");
    for (Mixer.Info info : AudioSystem.getMixerInfo()) {
      Mixer mixer = AudioSystem.getMixer(info);
      boolean hassrcline = false;
      for (Line.Info linfo : mixer.getSourceLineInfo())
        if (linfo instanceof javax.sound.sampled.DataLine.Info)
          hassrcline = true;
      if (hassrcline) {
        dev_list.add(info.getName());
      }
    }

    String[] devlist = new String[dev_list.size()];
    dev_list.toArray(devlist);

    JPanel panel = new JPanel();
    panel.setLayout(new BorderLayout());
    setContentPane(panel);

    JPanel optpanel = new JPanel();
    optpanel.setBorder(BorderFactory.createEmptyBorder(3, 3, 3, 3));
    panel.add(optpanel);
    optpanel.setLayout(new GridBagLayout());

    GridBagConstraints c = new GridBagConstraints();
    c.insets = new Insets(3, 3, 3, 3);
    c.anchor = GridBagConstraints.WEST;

    final JComboBox co_devname = new JComboBox(devlist);
```

```
    final JComboBox co_samplerate = new JComboBox(new String[] { "44100",
        "22050", "11025" });
    co_samplerate.setSelectedIndex(0);
    co_samplerate.setEditable(true);
    final JComboBox co_channels = new JComboBox(new String[] { "1", "2" });
    co_channels.setSelectedIndex(1);
    final JComboBox co_bits = new JComboBox(new String[] { "8", "16" });
    co_bits.setSelectedIndex(1);
    final JComboBox co_latency = new JComboBox(new String[] { "100", "200",
        "400", "800" });
    co_latency.setSelectedIndex(2);
    co_latency.setEditable(true);
    final JComboBox co_polyphony = new JComboBox(new String[] { "32", "64",
        "96", "128", "256" });
    co_polyphony.setSelectedIndex(1);
    co_polyphony.setEditable(true);
    final JComboBox co_interp = new JComboBox(new String[] { "linear",
        "cubic", "sinc", "point" });
    co_interp.setSelectedIndex(0);

        final JComboBox co_largemode = new JComboBox(new String[] { "true",
                "false" });
        co_largemode.setSelectedIndex(1);

    c.gridy = 0;
    c.gridx = 0;
    optpanel.add(new JLabel("Device name:"), c);
    c.gridy = 0;
    c.gridx = 1;
    optpanel.add(co_devname, c);
    c.gridy = 1;
    c.gridx = 0;
    optpanel.add(new JLabel("Sample rate (Hz):"), c);
    c.gridy = 1;
    c.gridx = 1;
    optpanel.add(co_samplerate, c);
    c.gridy = 2;
    c.gridx = 0;
    optpanel.add(new JLabel("Channels:"), c);
    c.gridy = 2;
    c.gridx = 1;
    optpanel.add(co_channels, c);
    c.gridy = 3;
    c.gridx = 0;
    optpanel.add(new JLabel("Bits:"), c);
    c.gridy = 3;
    c.gridx = 1;
    optpanel.add(co_bits, c);
    c.gridy = 4;
    c.gridx = 0;
    optpanel.add(new JLabel("Latency (msec):"), c);
    c.gridy = 4;
    c.gridx = 1;
    optpanel.add(co_latency, c);
    c.gridy = 5;
    c.gridx = 0;
    optpanel.add(new JLabel("Max polyphony:"), c);
    c.gridy = 5;
    c.gridx = 1;
    optpanel.add(co_polyphony, c);
    c.gridy = 6;
    c.gridx = 0;
```

```
185     optpanel.add(new JLabel("Interpolation mode:"), c);
186     c.gridy = 6;
187     c.gridx = 1;
188     optpanel.add(co_interp, c);
189         c.gridy = 7;
190         c.gridx = 0;
191         optpanel.add(new JLabel("Large mode:"), c);
192         c.gridy = 7;
193         c.gridx = 1;
194         optpanel.add(co_largemode, c);
195 
196     JButton okbutton = new JButton("OK");
197     okbutton.addActionListener(new ActionListener() {
198       public void actionPerformed(ActionEvent e) {
199 
200         Properties p = SimpleMidiPlayer.getConfig();
201 
202         if (co_devname.getSelectedIndex() == 0)
203           p.remove("devicename");
204         else
205           p.setProperty("devicename", getValueFromList(co_devname));
206 
207         p.setProperty("samplerate", getValueFromList(co_samplerate));
208         p.setProperty("channels", getValueFromList(co_channels));
209         p.setProperty("bits", getValueFromList(co_bits));
210         p.setProperty("latency", getValueFromList(co_latency));
211         p.setProperty("polyphony", getValueFromList(co_polyphony));
212                 p.setProperty("interpolation", getValueFromList(co_interp));
213         p.setProperty("largemode", getValueFromList(co_largemode));
214         SimpleMidiPlayer.storeConfig(p);
215         isok = true;
216 
217         ConfigDialog.this.dispose();
218 
219       }
220     });
221     okbutton.setDefaultCapable(true);
222     JButton cancelbutton = new JButton("Cancel");
223     cancelbutton.addActionListener(new ActionListener() {
224       public void actionPerformed(ActionEvent e) {
225         ConfigDialog.this.dispose();
226       }
227     });
228 
229     JPanel buttonpanel = new JPanel();
230     panel.add(buttonpanel, BorderLayout.SOUTH);
231     buttonpanel.add(okbutton);
232     buttonpanel.add(cancelbutton);
233 
234     KeyStroke stroke = KeyStroke.getKeyStroke(KeyEvent.VK_ESCAPE, 0);
235     panel.registerKeyboardAction(new ActionListener() {
236       public void actionPerformed(ActionEvent e) {
237         dispose();
238       }
239     }, stroke, JComponent.WHEN_IN_FOCUSED_WINDOW);
240 
241     Properties p = SimpleMidiPlayer.getConfig();
242 
243     selectValueInList(co_devname, p.getProperty("devicename"));
244     selectValueInList(co_samplerate, p.getProperty("samplerate"));
245     selectValueInList(co_channels, p.getProperty("channels"));
246     selectValueInList(co_bits, p.getProperty("bits"));
```

```
247     selectValueInList(co_latency, p.getProperty("latency"));
248     selectValueInList(co_polyphony, p.getProperty("polyphony"));
249     selectValueInList(co_interp, p.getProperty("interpolation"));
250         selectValueInList(co_largemode, p.getProperty("largemode"));
251 
252     pack();
253 
254     SimpleMidiPlayer.centerWindow(this);
255   }
256 
257 }
```

120 simplemidiplayer/InfoFrame.java

```
/*
 * Copyright (c) 2007 by Karl Helgason
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * - Redistributions of source code must retain the above copyright notice,
 *   this list of conditions and the following disclaimer.
 * - Redistributions in binary form must reproduce the above copyright
 *   notice, this list of conditions and the following disclaimer in the
 *   documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package simplemidiplayer;

import java.awt.BorderLayout;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.io.UnsupportedEncodingException;

import javax.sound.midi.Instrument;
import javax.sound.midi.MetaMessage;
import javax.sound.midi.MidiChannel;
import javax.sound.midi.MidiEvent;
import javax.sound.midi.Patch;
import javax.sound.midi.Sequence;
import javax.sound.midi.ShortMessage;
import javax.sound.midi.Soundbank;
import javax.sound.midi.Track;
import javax.sound.midi.VoiceStatus;
import javax.swing.BorderFactory;
import javax.swing.ImageIcon;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTabbedPane;
import javax.swing.JTable;
import javax.swing.KeyStroke;
import javax.swing.table.DefaultTableModel;

public class InfoFrame extends JFrame {
```

```
61
62  private static final long serialVersionUID = 1L;
63
64  SimpleMidiPlayer midiplayer;
65
66  DefaultTableModel seqmodel;
67
68  DefaultTableModel sbkmodel;
69
70  DefaultTableModel chmodel;
71
72  DefaultTableModel vocmodel;
73
74  Sequence seq = null;
75
76  Soundbank sbk = null;
77
78  JTabbedPane tabs;
79
80  JPanel seqtab;
81
82  JPanel sbktab;
83
84  JLabel sbkinfolab;
85
86  JPanel chtab;
87
88  JPanel voctab;
89
90  public InfoFrame(SimpleMidiPlayer midiplayer) {
91
92    this.midiplayer = midiplayer;
93    JPanel panel = new JPanel();
94    panel.setLayout(new BorderLayout());
95    panel.setBorder(BorderFactory.createEmptyBorder(3, 3, 3, 3));
96    setContentPane(panel);
97
98    KeyStroke stroke = KeyStroke.getKeyStroke(KeyEvent.VK_ESCAPE, 0);
99    panel.registerKeyboardAction(new ActionListener() {
100     public void actionPerformed(ActionEvent e) {
101       setVisible(false);
102     }
103   }, stroke, JComponent.WHEN_IN_FOCUSED_WINDOW);
104
105   tabs = new JTabbedPane();
106   panel.add(tabs);
107
108   seqtab = new JPanel();
109   seqtab.setLayout(new BorderLayout());
110   JTable seqtable = new JTable();
111
112   seqmodel = new DefaultTableModel() {
113     private static final long serialVersionUID = 1L;
114
115     public boolean isCellEditable(int row, int column) {
116       return false;
117     }
118   };
119   seqmodel.addColumn("Track");
120   seqmodel.addColumn("Channel");
121   seqmodel.addColumn("Patch");
122   seqmodel.addColumn("Instrument");
```

```
123     seqmodel.addColumn("Name");
124     seqtable.setModel(seqmodel);
125     seqtable.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
126     seqtable.getColumnModel().getColumn(0).setPreferredWidth(65);
127     seqtable.getColumnModel().getColumn(1).setPreferredWidth(65);
128     seqtable.getColumnModel().getColumn(2).setPreferredWidth(65);
129     seqtable.getColumnModel().getColumn(3).setPreferredWidth(100);
130     seqtable.getColumnModel().getColumn(4).setPreferredWidth(200);
131 
132     seqtab.add(new JScrollPane(seqtable));
133     seqtab.setOpaque(false);
134     tabs.addTab("Sequence", seqtab);
135 
136     sbktab = new JPanel();
137     sbkinfolab = new JLabel();
138     sbkinfolab.setFont(sbkinfolab.getFont().deriveFont(Font.PLAIN));
139     sbkinfolab.setBorder(BorderFactory.createEmptyBorder(2, 2, 2, 2));
140     sbktab.setLayout(new BorderLayout());
141     sbktab.add(sbkinfolab, BorderLayout.NORTH);
142     JTable sbktable = new JTable();
143 
144     sbkmodel = new DefaultTableModel() {
145       private static final long serialVersionUID = 1L;
146 
147       public boolean isCellEditable(int row, int column) {
148         return false;
149       }
150     };
151 
152     sbkmodel.addColumn("Patch");
153     sbkmodel.addColumn("Name");
154     sbkmodel.addColumn("Type");
155     sbktable.setModel(sbkmodel);
156     sbktable.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
157     sbktable.getColumnModel().getColumn(0).setPreferredWidth(80);
158     sbktable.getColumnModel().getColumn(1).setPreferredWidth(200);
159     sbktable.getColumnModel().getColumn(2).setPreferredWidth(200);
160 
161     sbktab.add(new JScrollPane(sbktable));
162     sbktab.setOpaque(false);
163     tabs.addTab("Soundbank", sbktab);
164 
165     chtab = new JPanel();
166     chtab.setLayout(new BorderLayout());
167     JTable chtable = new JTable();
168 
169     chmodel = new DefaultTableModel() {
170       private static final long serialVersionUID = 1L;
171 
172       public boolean isCellEditable(int row, int column) {
173         return false;
174       }
175     };
176 
177     chmodel.addColumn("Channel");
178     chmodel.addColumn("Instrument");
179     chmodel.addColumn("Pitch");
180     chmodel.addColumn("Volume");
181     chmodel.addColumn("Pan");
182     chmodel.addColumn("Reverb");
183     chmodel.addColumn("Chorus");
184     chtable.setModel(chmodel);
```

```
    chtable.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
    chtable.getColumnModel().getColumn(0).setPreferredWidth(65);
    chtable.getColumnModel().getColumn(1).setPreferredWidth(100);
    chtable.getColumnModel().getColumn(2).setPreferredWidth(65);
    chtable.getColumnModel().getColumn(3).setPreferredWidth(65);
    chtable.getColumnModel().getColumn(4).setPreferredWidth(65);
    chtable.getColumnModel().getColumn(5).setPreferredWidth(65);
    chtable.getColumnModel().getColumn(6).setPreferredWidth(65);
    chmodel.addRow(new Object[] { "1", "", 0, 100, 0, 64, 0 });
    chmodel.addRow(new Object[] { "2", "", 0, 100, 0, 64, 0 });
    chmodel.addRow(new Object[] { "3", "", 0, 100, 0, 64, 0 });
    chmodel.addRow(new Object[] { "4", "", 0, 100, 0, 64, 0 });
    chmodel.addRow(new Object[] { "5", "", 0, 100, 0, 64, 0 });
    chmodel.addRow(new Object[] { "6", "", 0, 100, 0, 64, 0 });
    chmodel.addRow(new Object[] { "7", "", 0, 100, 0, 64, 0 });
    chmodel.addRow(new Object[] { "8", "", 0, 100, 0, 64, 0 });
    chmodel.addRow(new Object[] { "9", "", 0, 100, 0, 64, 0 });
    chmodel.addRow(new Object[] { "10", "", 0, 100, 0, 64, 0 });
    chmodel.addRow(new Object[] { "11", "", 0, 100, 0, 64, 0 });
    chmodel.addRow(new Object[] { "12", "", 0, 100, 0, 64, 0 });
    chmodel.addRow(new Object[] { "13", "", 0, 100, 0, 64, 0 });
    chmodel.addRow(new Object[] { "14", "", 0, 100, 0, 64, 0 });
    chmodel.addRow(new Object[] { "15", "", 0, 100, 0, 64, 0 });
    chmodel.addRow(new Object[] { "16", "", 0, 100, 0, 64, 0 });

    chtab.add(new JScrollPane(chtable));
    chtab.setOpaque(false);
    tabs.addTab("Channels", chtab);

    voctab = new JPanel();
    voctab.setLayout(new BorderLayout());
    JTable voctable = new JTable();

    vocmodel = new DefaultTableModel() {
      private static final long serialVersionUID = 1L;

      public boolean isCellEditable(int row, int column) {
        return false;
      }
    };
    vocmodel.addColumn("Active");
    vocmodel.addColumn("Channel");
    vocmodel.addColumn("Bank");
    vocmodel.addColumn("Program");
    vocmodel.addColumn("Note");
    vocmodel.addColumn("Volume");
    voctable.setModel(vocmodel);
    voctable.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
    voctable.getColumnModel().getColumn(0).setPreferredWidth(65);
    voctable.getColumnModel().getColumn(1).setPreferredWidth(64);
    voctable.getColumnModel().getColumn(2).setPreferredWidth(65);
    voctable.getColumnModel().getColumn(3).setPreferredWidth(65);
    voctable.getColumnModel().getColumn(4).setPreferredWidth(65);
    voctable.getColumnModel().getColumn(5).setPreferredWidth(65);

    voctab.add(new JScrollPane(voctable));
    voctab.setOpaque(false);
    tabs.addTab("Voices", voctab);

    ImageIcon swan = new javax.swing.ImageIcon(getClass().getResource(
        "/simplemidiplayer/swan.png"));
    setIconImage(swan.getImage());
```

```
247     setTitle("Gervill - MIDI Player - Info");
248     setSize(550, 350);
249     // setLocationByPlatform(true);
250 
251     SimpleMidiPlayer.centerWindow(this);
252 
253   }
254 
255   public void updateDisplay() {
256     if (!isVisible())
257       return;
258     if (!midiplayer.player_running)
259       return;
260 
261     if (tabs.getSelectedComponent() == seqtab) {
262       if (midiplayer.seq != seq) {
263         seq = midiplayer.seq;
264         while (seqmodel.getRowCount() != 0)
265           seqmodel.removeRow(0);
266 
267         int row = 0;
268         for (Track track : seq.getTracks()) {
269 
270           int channel = 0;
271           int program = 0;
272           int bank_lsb = -1;
273           int bank_msb = -1;
274           String instext = "";
275           String tracktext = "";
276 
277           int evcount = track.size();
278           for (int i = 0; i < evcount; i++) {
279             MidiEvent event = track.get(i);
280             if (event.getTick() != 0)
281               break;
282             if (event.getMessage() instanceof MetaMessage) {
283               MetaMessage mmsg = (MetaMessage) event.getMessage();
284               try {
285                 if (mmsg.getType() == 3)
286                   tracktext = new String(mmsg.getData(),
287                       "Latin1");
288                 if (mmsg.getType() == 4)
289                   instext = new String(mmsg.getData(),
290                       "Latin1");
291               } catch (UnsupportedEncodingException e) {
292               }
293             }
294             if (event.getMessage() instanceof ShortMessage) {
295               ShortMessage smsg = (ShortMessage) event
296                   .getMessage();
297               channel = smsg.getChannel() + 1;
298 
299               if (smsg.getCommand() == ShortMessage.PROGRAM_CHANGE)
300                 program = smsg.getData1();
301               if (smsg.getCommand() == ShortMessage.CONTROL_CHANGE) {
302                 if (smsg.getData1() == 0)
303                   bank_msb = smsg.getData2();
304                 if (smsg.getData1() == 32)
305                   bank_lsb = smsg.getData2();
306               }
307 
308             }
```

```
309
310         }
311
312         String[] rowdata = new String[5];
313
314         if (instext.length() == 0)
315           if (midiplayer.sbk != null) {
316             int bank = 0;
317             if (bank_msb != -1)
318               bank += bank_msb * 128;
319             if (bank_lsb != -1)
320               bank += bank_lsb;
321             Patch patch = new Patch(bank, program);
322             Instrument ins = midiplayer.sbk
323                 .getInstrument(patch);
324             if (ins != null)
325               instext = ins.getName();
326           }
327
328         rowdata[0] = "" + row;
329         rowdata[1] = "" + channel;
330         rowdata[2] = "0," + program;
331         rowdata[3] = instext;
332         rowdata[4] = tracktext;
333
334         if (bank_msb != -1 || bank_lsb != -1) {
335           if (bank_msb == -1)
336             bank_msb = 0;
337           if (bank_lsb == -1)
338             bank_lsb = 0;
339           rowdata[2] = (bank_msb * 128 + bank_lsb) + ","
340               + program;
341         }
342
343         seqmodel.addRow(rowdata);
344         row++;
345       }
346
347     }
348   }
349
350   if (tabs.getSelectedComponent() == sbktab) {
351     if (midiplayer.sbk != sbk)
352       if (midiplayer.sbk != null) {
353         sbk = midiplayer.sbk;
354         while (sbkmodel.getRowCount() != 0)
355           sbkmodel.removeRow(0);
356
357         sbkinfolab.setText("<html><body><table>"
358             + "<tr><td><b>Name:</b></td><td>" + sbk.getName()
359             + "</td>" + "<td><b>␣␣Description:</b></td><td>"
360             + sbk.getDescription() + "</td></tr>"
361             + "<tr><td><b>Version:</b></td><td>"
362             + sbk.getVersion() + "</td>"
363             + "<td><b>␣␣Vendor:</b></td><td>" + sbk.getVendor()
364             + "</td></tr></table>");
365
366         for (Instrument ins : sbk.getInstruments()) {
367           String[] rowdata = new String[3];
368           rowdata[0] = ins.getPatch().getBank() + ","
369               + ins.getPatch().getProgram();
370           rowdata[1] = ins.getName();
```

```
371             rowdata[2] = ins.getClass().getSimpleName();
372             sbkmodel.addRow(rowdata);
373           }
374         }
375     }
376
377     if (tabs.getSelectedComponent() == chtab) {
378       Soundbank sbk = midiplayer.sbk;
379       MidiChannel[] channels = midiplayer.softsynth.getChannels();
380       for (int i = 0; i < 16; i++) {
381         MidiChannel channel = channels[i];
382         if (sbk != null) {
383           Patch patch = new Patch(channel.getController(0) * 128
384               + channel.getController(32), channel.getProgram());
385           Instrument ins = sbk.getInstrument(patch);
386           if (ins != null)
387             chmodel.setValueAt(channel.getProgram() + ": "
388                 + ins.getName(), i, 1);
389         }
390
391         chmodel.setValueAt(channel.getPitchBend() - 8192, i, 2);
392         chmodel.setValueAt(channel.getController(7), i, 3);
393         chmodel.setValueAt(channel.getController(10) - 64, i, 4);
394         chmodel.setValueAt(channel.getController(91), i, 5);
395         chmodel.setValueAt(channel.getController(93), i, 6);
396       }
397     }
398
399     if (tabs.getSelectedComponent() == voctab) {
400       {
401         VoiceStatus[] voices = midiplayer.softsynth.getVoiceStatus();
402         while (vocmodel.getRowCount() > voices.length)
403           vocmodel.removeRow(vocmodel.getRowCount() - 1);
404         while (vocmodel.getRowCount() < voices.length)
405           vocmodel.addRow(new Object[] { false, 0, 0, 0, 0, 0 });
406         for (int i = 0; i < voices.length; i++) {
407           VoiceStatus voc = voices[i];
408           vocmodel.setValueAt(voc.active, i, 0);
409           vocmodel.setValueAt(voc.channel + 1, i, 1);
410           vocmodel.setValueAt(voc.bank, i, 2);
411           vocmodel.setValueAt(voc.program, i, 3);
412           vocmodel.setValueAt(voc.note, i, 4);
413           vocmodel.setValueAt(voc.volume, i, 5);
414         }
415       }
416     }
417   }
418 }
```

121 simplemidiplayer/SimpleMidiPlayer.java

```java
/*
 * Copyright (c) 2007 by Karl Helgason
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * - Redistributions of source code must retain the above copyright notice,
 *   this list of conditions and the following disclaimer.
 * - Redistributions in binary form must reproduce the above copyright
 *   notice, this list of conditions and the following disclaimer in the
 *   documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package simplemidiplayer;

import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.GraphicsConfiguration;
import java.awt.GraphicsEnvironment;
import java.awt.Insets;
import java.awt.Point;
import java.awt.Rectangle;
import java.awt.Toolkit;
import java.awt.Window;
import java.awt.datatransfer.DataFlavor;
import java.awt.datatransfer.Transferable;
import java.awt.datatransfer.UnsupportedFlavorException;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Properties;

import javax.sound.midi.InvalidMidiDataException;
import javax.sound.midi.MidiChannel;
import javax.sound.midi.MidiDevice;
```

```
61 import javax.sound.midi.MidiSystem;
62 import javax.sound.midi.MidiUnavailableException;
63 import javax.sound.midi.Sequence;
64 import javax.sound.midi.Sequencer;
65 import javax.sound.midi.Soundbank;
66 import javax.sound.midi.Synthesizer;
67 import javax.sound.midi.SysexMessage;
68 import javax.sound.midi.MidiDevice.Info;
69 import javax.sound.sampled.AudioFormat;
70 import javax.sound.sampled.AudioSystem;
71 import javax.sound.sampled.DataLine;
72 import javax.sound.sampled.Line;
73 import javax.sound.sampled.Mixer;
74 import javax.sound.sampled.SourceDataLine;
75 import javax.swing.Icon;
76 import javax.swing.ImageIcon;
77 import javax.swing.JButton;
78 import javax.swing.JComponent;
79 import javax.swing.JFileChooser;
80 import javax.swing.JFrame;
81 import javax.swing.JLabel;
82 import javax.swing.JMenuItem;
83 import javax.swing.JPanel;
84 import javax.swing.JPopupMenu;
85 import javax.swing.SwingUtilities;
86 import javax.swing.TransferHandler;
87 import javax.swing.filechooser.FileFilter;
88
89 import com.sun.media.sound.AudioSynthesizer;
90 import com.sun.media.sound.EmergencySoundbank;
91
92 public class SimpleMidiPlayer extends JFrame {
93
94   public class ImagePanel extends JPanel {
95
96     private static final long serialVersionUID = 1L;
97
98     Icon icon;
99
100    public ImagePanel(Icon icon) {
101      super();
102      this.icon = icon;
103    }
104
105    protected void paintComponent(Graphics g) {
106      super.paintComponent(g);
107      icon.paintIcon(this, g, 0, 0);
108    }
109
110  }
111
112  private static final long serialVersionUID = 1L;
113
114  public static void main(String[] args) {
115    if (!configExists()) {
116      ConfigDialog cd = new ConfigDialog(null);
117      cd.setVisible(true);
118      if (!cd.isOK())
119        return;
120    }
121    new SimpleMidiPlayer().setVisible(true);
122  }
```

```
123
124   public JButton makeButton(String caption) {
125     JButton butt = new JButton(caption);
126     butt.setMargin(new Insets(2, 2, 2, 2));
127     butt.setFocusable(false);
128     butt.setFont(butt.getFont().deriveFont(Font.PLAIN));
129     return butt;
130   }
131
132   JPopupMenu loadmenu;
133
134   boolean synth_loaded = false;
135
136   Sequencer seqr = null;
137
138   Sequence seq = null;
139
140   String seq_errmsg = null;
141
142   File seqfile = null;
143
144   Soundbank sbk = null;
145
146   String sbk_errmsg = null;
147
148   File sbkfile = null;
149
150   Synthesizer softsynth = null;
151
152   Mixer synthmixer = null;
153   SourceDataLine line = null;
154
155   InfoFrame infoframe;
156
157   AudioFormat format;
158
159   /*
160    * Find available AudioSynthesizer.
161    */
162   public static AudioSynthesizer findAudioSynthesizer()
163       throws MidiUnavailableException {
164     // First check if default synthesizer is AudioSynthesizer.
165     Synthesizer synth = MidiSystem.getSynthesizer();
166     if (synth instanceof AudioSynthesizer)
167       return (AudioSynthesizer) synth;
168
169     // If default synhtesizer is not AudioSynthesizer, check others.
170     Info[] infos = MidiSystem.getMidiDeviceInfo();
171     for (int i = 0; i < infos.length; i++) {
172       MidiDevice dev = MidiSystem.getMidiDevice(infos[i]);
173       if (dev instanceof AudioSynthesizer)
174         return (AudioSynthesizer) dev;
175     }
176
177     // No AudioSynthesizer was found, return null.
178     return null;
179   }
180
181   public void initMIDI() {
182     try {
183
184       final AudioSynthesizer synth = findAudioSynthesizer();
```

```
185
186       Properties p = getConfig();
187       Map<String, Object> ainfo = new HashMap<String, Object>();
188
189       try {
190
191         format = new AudioFormat(Float.parseFloat(p
192             .getProperty("samplerate", "44100")), Integer
193             .parseInt(p.getProperty("bits", "16")), Integer
194             .parseInt(p.getProperty("channels", "2")), true, false);
195
196         int latency = Integer.parseInt(p.getProperty("latency", "200")) * 1000;
197
198         String devname = p.getProperty("devicename");
199         if (devname != null) {
200           Mixer.Info selinfo = null;
201           for (Mixer.Info info : AudioSystem.getMixerInfo()) {
202             Mixer mixer = AudioSystem.getMixer(info);
203             boolean hassrcline = false;
204             for (Line.Info linfo : mixer.getSourceLineInfo())
205               if (linfo instanceof javax.sound.sampled.DataLine.Info)
206                 hassrcline = true;
207             if (hassrcline) {
208               if (info.getName().equals(devname)) {
209                 selinfo = info;
210                 break;
211               }
212             }
213           }
214           if (selinfo != null) {
215             synthmixer = AudioSystem.getMixer(selinfo);
216             try {
217               synthmixer.open();
218
219               int bufferSize = (int)
220                 (format.getFrameSize() * format.getFrameRate()
221                 * latency / 1000000f);
222               if(bufferSize < 500) bufferSize = 500;
223
224               DataLine.Info dataLineInfo = new DataLine.Info(
225                   SourceDataLine.class, format, bufferSize);
226               if (synthmixer.isLineSupported(dataLineInfo))
227                 line = (SourceDataLine) synthmixer
228                     .getLine(dataLineInfo);
229
230               line.open(format, bufferSize);
231               line.start();
232
233             } catch (Throwable t) {
234               t.printStackTrace();
235               synthmixer = null;
236             }
237           }
238         }
239
240         //ainfo.put("multi threading", true);
241         ainfo.put("format", format);
242         ainfo.put("max␣polyphony", Integer.parseInt(p.getProperty(
243             "polyphony", "64")));
244         ainfo.put("latency", Long.parseLong(p.getProperty("latency",
245             "200")) * 1000L);
246
```

```
247         ainfo.put("interpolation", p.getProperty("interpolation"));
248                 String largemode = p.getProperty("largemode");
249                 if(largemode == null) largemode = "false";
250                 ainfo.put("large_mode", largemode.equalsIgnoreCase("true"));
251 
252       } catch (Throwable t) {
253         t.printStackTrace();
254       }
255 
256       synth.open(line, ainfo);
257 
258       Runnable r = new Runnable() {
259         public void run() {
260           softsynth = synth;
261           if (sbk == null)
262             sbk = synth.getDefaultSoundbank();
263           try {
264             if (seqr == null) {
265               try {
266                 seqr = MidiSystem.getSequencer(false);
267               } catch (MidiUnavailableException e2) {
268                 e2.printStackTrace();
269               }
270             }
271             if (seqr.isOpen())
272               seqr.close();
273             seqr.getTransmitter().setReceiver(
274                 softsynth.getReceiver());
275             seqr.open();
276           } catch (MidiUnavailableException e) {
277             e.printStackTrace();
278           }
279           synth_loaded = true;
280         }
281       };
282 
283       if (SwingUtilities.isEventDispatchThread())
284         r.run();
285       else
286         SwingUtilities.invokeLater(r);
287     } catch (Exception e) {
288       e.printStackTrace();
289     }
290   }
291 
292   public void initMIDI_inThread() {
293     synth_loaded = false;
294     new Thread() {
295       public void run() {
296         initMIDI();
297       }
298     }.start();
299 
300   }
301 
302   public void closeMIDI() {
303     if (synth_loaded) {
304       seqr.close();
305       softsynth.close();
306       if (line != null) {
307         line.close();
308         line = null;
```

```
309       }
310       if (synthmixer != null) {
311         synthmixer.close();
312         synthmixer = null;
313       }
314     }
315   }
316
317   JLabel displayLab = new JLabel();
318
319   public void updateDisplay() {
320
321     if (!synth_loaded) {
322       displayLab.setText("<html><body>Initializing . . . .");
323     } else {
324       MidiDevice.Info info = softsynth.getDeviceInfo();
325
326       String fmts = (int) format.getSampleRate() + "Hz "
327           + format.getSampleSizeInBits() + "bit "
328           + format.getChannels() + "ch";
329       String line1 = "<b>" + info.getName() + " " + info.getVersion()
330           + "</b>  " + fmts;
331       String line2 = "";
332
333       if (sbk == null) {
334         line2 = "No SoundBank Loaded!";
335       } else {
336         if (sbk_errmsg != null)
337           line2 = sbk_errmsg;
338         else if (sbkfile == null)
339           line2 = "Default SoundBank";
340         else
341           line2 = sbkfile.getName();
342         if (line2.length() > 31)
343           line2 = line2.substring(0, 31);
344       }
345
346       String line3 = "";
347       if (seq == null) {
348         line3 = "No Sequence";
349       } else {
350         if (seqr.isRunning() || seqr.getTickPosition() != 0) {
351
352           long a = seqr.getTickPosition() / seq.getResolution();
353           long b = seqr.getTickLength() / seq.getResolution();
354           if (seqr.isRunning())
355             line3 = "PLAY " + a + " of " + b;
356           else
357             line3 = "STOP " + a + " of " + b;
358
359         } else {
360           if (seq_errmsg != null)
361             line3 = seq_errmsg;
362           else
363             line3 = seqfile.getName();
364           if (line3.length() > 31)
365             line3 = line3.substring(0, 31);
366         }
367       }
368       displayLab.setText("<html><body>" + line1 + "<br>" + line2 + "<br>"
369           + line3);
370     }
```

```
371
372   }
373
374   JFileChooser loadseq;
375
376   JFileChooser loadsndbk;
377
378   Thread actdisplay;
379
380   boolean player_running = true;
381
382   private static String CONFIG_FILE_NAME = "SimpleMidiPlayer.xml";
383
384   private static File userDir = new File(System.getProperty("user.home"),
385       ".gervill");
386
387   private static File configFile = new File(userDir, CONFIG_FILE_NAME);
388
389   private static Properties configp = null;
390
391   public static void centerWindow(Window w) {
392     Rectangle windowSize;
393     // Insets windowInsets;
394
395     Toolkit toolkit = Toolkit.getDefaultToolkit();
396     GraphicsEnvironment ge = java.awt.GraphicsEnvironment
397         .getLocalGraphicsEnvironment();
398     GraphicsConfiguration gc = ge.getDefaultScreenDevice()
399         .getDefaultConfiguration();
400     if (gc == null)
401       gc = w.getGraphicsConfiguration();
402
403     if (gc != null) {
404       windowSize = gc.getBounds();
405     } else {
406       windowSize = new java.awt.Rectangle(toolkit.getScreenSize());
407     }
408
409     Dimension size = w.getSize();
410     Point parent_loc = w.getLocation();
411     w.setLocation(parent_loc.x + windowSize.width / 2 - (size.width / 2),
412         parent_loc.y + windowSize.height / 2 - (size.height / 2));
413
414   }
415
416   public static boolean configExists() {
417     synchronized (configFile) {
418       return configFile.exists();
419     }
420   }
421
422   public static Properties getConfig() {
423     synchronized (configFile) {
424
425       if (configp != null) {
426         Properties p = new Properties();
427         p.putAll(configp);
428         return p;
429       }
430       Properties p = new Properties();
431       if (configFile.exists()) {
432         FileInputStream fis;
```

```
433         try {
434           fis = new FileInputStream(configFile);
435           try {
436             p.loadFromXML(fis);
437           } finally {
438             fis.close();
439           }
440         } catch (Exception e) {
441           e.printStackTrace();
442         }
443       }
444       return p;
445 
446     }
447   }
448 
449   public static void storeConfig(Properties p) {
450     synchronized (configFile) {
451 
452       try {
453         configp = new Properties();
454         configp.putAll(p);
455 
456         if (!userDir.exists())
457           userDir.mkdirs();
458         FileOutputStream fos = new FileOutputStream(configFile);
459         try {
460           p.storeToXML(fos, "GervillMidiPlayer");
461         } finally {
462           fos.close();
463         }
464       } catch (Exception e) {
465         e.printStackTrace();
466       }
467 
468     }
469   }
470 
471   public void loadMidiSeq(File newseqfile) {
472     try {
473       seq_errmsg = null;
474       Sequence newseq = MidiSystem.getSequence(newseqfile);
475 
476       seq = newseq;
477       seqfile = newseqfile;
478       // boolean running = seqr.isRunning();
479       seqr.stop();
480 
481             // Reset All Channels
482             for(MidiChannel c : softsynth.getChannels())
483                 c.resetAllControllers();
484 
485       seqr.setSequence(seq);
486       seqr.setTickPosition(0);
487       seqr.start();
488     } catch (Throwable e1) {
489       seq_errmsg = e1.toString();
490     }
491 
492   }
493 
494   public void loadSoundbank(File newsbkfile) {
```

```
    try {
      sbk_errmsg = null;
      Soundbank newsbk = MidiSystem.getSoundbank(newsbkfile);
      if (sbk != null)
        softsynth.unloadAllInstruments(sbk);
      sbkfile = newsbkfile;
      sbk = newsbk;
      softsynth.loadAllInstruments(sbk);
    } catch (Throwable e1) {
      sbk_errmsg = e1.toString();
    }

  }

  public SimpleMidiPlayer() {

    loadseq = new JFileChooser();
    loadseq.setDialogTitle("Load MIDI Sequence");
    loadseq.setFileFilter(new FileFilter() {
      public boolean accept(File f) {
        if (!f.isFile())
          return true;
        return f.getName().toLowerCase().endsWith(".mid");
      }

      public String getDescription() {
        return "MIDI Sequence";
      }
    });
    loadsndbk = new JFileChooser();
    loadsndbk.setFileFilter(new FileFilter() {
      public boolean accept(File f) {
        if (!f.isFile())
          return true;
        String name = f.getName().toLowerCase();
        if (name.endsWith(".sf2"))
          return true;
        if (name.endsWith(".dls"))
          return true;
        if (name.endsWith(".pat"))
          return true;
        if (name.endsWith(".cfg"))
          return true;
        if (name.endsWith(".wav"))
          return true;
        if (name.endsWith(".au"))
          return true;
        if (name.endsWith(".aif"))
          return true;
        return false;
      }

      public String getDescription() {
        return "SoundBank (*.sf2,*.dls,*.pat,*.cfg,*.wav,*.au,*.aif)";
      }
    });

    // setLocationByPlatform(true);
    setResizable(false);
    setTitle("Gervill - MIDI Player");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

```

```
    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        closeMIDI();
        player_running = false;
        try {
          actdisplay.join(1000);
        } catch (InterruptedException e1) {
          e1.printStackTrace();
        }
      }
    });

    infoframe = new InfoFrame(this);

    actdisplay = new Thread() {
      public void run() {
        boolean ok = true;
        while (ok) {
          synchronized (SimpleMidiPlayer.this) {
            ok = player_running;
          }
          SwingUtilities.invokeLater(new Runnable() {
            public void run() {
              updateDisplay();
              infoframe.updateDisplay();
            }
          });
          try {
            Thread.sleep(100);
          } catch (InterruptedException e) {
            e.printStackTrace();
            return;
          }
        }
      }
    };
    actdisplay.start();

    initMIDI_inThread();

    ImageIcon backgr = new javax.swing.ImageIcon(getClass().getResource(
        "/simplemidiplayer/backgr.png"));
    ImageIcon swan = new javax.swing.ImageIcon(getClass().getResource(
        "/simplemidiplayer/swan.png"));
    setIconImage(swan.getImage());

    JPanel panel = new ImagePanel(backgr);
    Dimension size = new Dimension(443, 125);
    panel.setPreferredSize(size);
    panel.setMinimumSize(size);
    panel.setLayout(null);

    TransferHandler thandler = new TransferHandler() {
      private static final long serialVersionUID = 1L;

      public boolean canImport(JComponent comp,
          DataFlavor[] transferFlavors) {

        for (int i = 0; i < transferFlavors.length; i++) {
          if (transferFlavors[i]
              .equals(DataFlavor.javaFileListFlavor)) {
            return true;
```

```
619           }
620         }
621         return false;
622       }
623
624       public boolean importData(JComponent comp, Transferable t) {
625
626         List files = null;
627         try {
628           files = (List) t
629               .getTransferData(DataFlavor.javaFileListFlavor);
630         } catch (UnsupportedFlavorException e) {
631           e.printStackTrace();
632         } catch (IOException e) {
633           e.printStackTrace();
634         }
635
636         if (files == null)
637           return false;
638
639         for (Object o : files) {
640           File file = (File) o;
641           if (file.isFile()) {
642             if (file.getName().toLowerCase().endsWith(".mid"))
643               loadMidiSeq(file);
644             else
645               loadSoundbank(file);
646           }
647         }
648
649         return true;
650       }
651     };
652
653     panel.setTransferHandler(thandler);
654
655     setContentPane(panel);
656
657     displayLab.setSize(225, 67);
658     displayLab.setLocation(206, 20);
659     displayLab.setFont(new Font("Monospaced", Font.PLAIN, 12));
660     displayLab.setVerticalAlignment(JLabel.TOP);
661     displayLab.setVerticalTextPosition(JLabel.TOP);
662     displayLab.setTransferHandler(thandler);
663     panel.add(displayLab);
664
665     JPanel toolBar = new JPanel();
666     toolBar.setLayout(new FlowLayout(FlowLayout.RIGHT, 2, 5));
667     toolBar.setSize(429, 80);
668     toolBar.setLocation(0, 82);
669     toolBar.setOpaque(false);
670
671     final JButton config = makeButton("CONFIG");
672     final JButton info = makeButton("INFO");
673     final JButton load = makeButton("LOAD");
674     final JButton play = makeButton("PLAY");
675     final JButton stop = makeButton("STOP");
676
677     JMenuItem loadseq_menuitem = new JMenuItem("MIDI␣Sequence...");
678
679     loadseq_menuitem.addActionListener(new ActionListener() {
680       public void actionPerformed(ActionEvent e) {
```

```
681         if (!synth_loaded)
682           return;
683         if (loadseq.showOpenDialog(SimpleMidiPlayer.this) == JFileChooser.APPROVE_OPTION) {
684           loadMidiSeq(loadseq.getSelectedFile());
685         }
686       }
687     });
688 
689     JMenuItem loadsndbk_menuitem = new JMenuItem("Soundbank...");
690 
691     loadsndbk_menuitem.addActionListener(new ActionListener() {
692       public void actionPerformed(ActionEvent e) {
693         if (!synth_loaded)
694           return;
695         if (loadsndbk.showOpenDialog(SimpleMidiPlayer.this) == JFileChooser.APPROVE_OPTION) {
696           loadSoundbank(loadsndbk.getSelectedFile());
697         }
698       }
699     });
700 
701     JMenuItem default_loadsndbk_menuitem = new JMenuItem(
702         "Default␣Soundbank");
703 
704     default_loadsndbk_menuitem.addActionListener(new ActionListener() {
705       public void actionPerformed(ActionEvent e) {
706         if (softsynth.getDefaultSoundbank() != null) {
707           if (sbk != null)
708             softsynth.unloadAllInstruments(sbk);
709           sbk = softsynth.getDefaultSoundbank();
710           sbkfile = null;
711           softsynth.loadAllInstruments(sbk);
712         }
713       }
714     });
715 
716     JMenuItem emerg_loadsndbk_menuitem = new JMenuItem(
717     "Emergency␣Soundbank");
718 
719     emerg_loadsndbk_menuitem.addActionListener(new ActionListener() {
720     public void actionPerformed(ActionEvent e) {
721 
722       Soundbank emsbk;
723       try {
724         emsbk = EmergencySoundbank.createSoundbank();
725       } catch (Exception e1) {
726         e1.printStackTrace();
727         return;
728       }
729       if (sbk != null)
730         softsynth.unloadAllInstruments(sbk);
731       sbk = emsbk;
732       sbkfile = null;
733       softsynth.loadAllInstruments(sbk);
734     }
735     });
736 
737     loadmenu = new JPopupMenu();
738     loadmenu.add(loadseq_menuitem);
739     loadmenu.addSeparator();
740     loadmenu.add(loadsndbk_menuitem);
741     loadmenu.add(default_loadsndbk_menuitem);
742     loadmenu.add(emerg_loadsndbk_menuitem);
```

```
743
744     config.addActionListener(new ActionListener() {
745       public void actionPerformed(ActionEvent e) {
746         if (!synth_loaded)
747           return;
748
749         ConfigDialog cd = new ConfigDialog(SimpleMidiPlayer.this);
750         cd.setVisible(true);
751         if (cd.isOK()) {
752           Sequence pseq = seqr.getSequence();
753           long ptick = seqr.getTickPosition();
754           boolean prunning = seqr.isRunning();
755           seqr.stop();
756           softsynth.close();
757           if (synthmixer != null) {
758             synthmixer.close();
759             synthmixer = null;
760           }
761           initMIDI();
762           if (pseq != null) {
763             try {
764               seqr.setSequence(pseq);
765             } catch (InvalidMidiDataException e1) {
766               e1.printStackTrace();
767             }
768           }
769           seqr.setTickPosition(ptick);
770           if (prunning) {
771             seqr.start();
772           }
773
774         }
775       }
776     });
777
778     load.addActionListener(new ActionListener() {
779       public void actionPerformed(ActionEvent e) {
780         if (!synth_loaded)
781           return;
782         loadmenu.show(load, 0, 0);
783       }
784     });
785
786     info.addActionListener(new ActionListener() {
787       public void actionPerformed(ActionEvent e) {
788         infoframe.setVisible(!infoframe.isVisible());
789       }
790     });
791
792     play.addActionListener(new ActionListener() {
793       public void actionPerformed(ActionEvent e) {
794         if (!synth_loaded)
795           return;
796         if (seq == null)
797           return;
798         seqr.start();
799       }
800     });
801
802     stop.addActionListener(new ActionListener() {
803       public void actionPerformed(ActionEvent e) {
804
```

```
805         if (!synth_loaded)
806           return;
807         if (seq == null)
808           return;
809         if (seqr.isRunning())
810           seqr.stop();
811         else
812           seqr.setTickPosition(0);
813       }
814     });
815 
816     toolBar.add(config);
817     toolBar.add(load);
818     toolBar.add(info);
819     toolBar.add(play);
820     toolBar.add(stop);
821     panel.add(toolBar);
822 
823     pack();
824 
825     centerWindow(this);
826 
827   }
828 
829 }
```