

Contents

1	AudioRender.java	6
2	FMTest1.java	7
3	GeneratedInstrument1.java	9
4	MakeSoundFont.java	35
5	Midi2WavRender.java	37
6	ReverbAndChorusApplet.java	41
7	SimpleApplet1.java	49
8	TuningApplet1.java	52
9	TuningApplet2.java	57
10	TuningApplet3.java	62
11	UniversalSysExBuilder.java	70
12	VirtualKeyboard12.java	81
13	VirtualKeyboard19.java	87
14	VirtualKeyboard5.java	94
15	VirtualKeyboard7.java	99
16	com/sun/media/sound/AudioFileSoundbankReader.java	104
17	com/sun/media/sound/AudioFloatConverter.java	107
18	com/sun/media/sound/AudioFloatFormatConverter.java	125
19	com/sun/media/sound/AudioFloatInputStream.java	135
20	com/sun/media/sound/AudioSynthesizer.java	140
21	com/sun/media/sound/AudioSynthesizerPropertyInfo.java	143
22	com/sun/media/sound/DLSInfo.java	145
23	com/sun/media/sound/DLSInstrument.java	147
24	com/sun/media/sound/DLSModulator.java	155
25	com/sun/media/sound/DLSRegion.java	161
26	com/sun/media/sound/DLSSample.java	164
27	com/sun/media/sound/DLSSampleLoop.java	166

28	com/sun/media/sound/DLSSampleOptions.java	168
29	com/sun/media/sound/DLSSoundbank.java	170
30	com/sun/media/sound/DLSSoundbankReader.java	191
31	com/sun/media/sound/EmergencySoundbank.java	193
32	com/sun/media/sound/FFT.java	237
33	com/sun/media/sound/InvalidDataException.java	250
34	com/sun/media/sound/InvalidFormatException.java	251
35	com/sun/media/sound/JARSoundbankReader.java	252
36	com/sun/media/sound/MidiDeviceReceiver.java	254
37	com/sun/media/sound/ModelAbstractChannelMixer.java	255
38	com/sun/media/sound/ModelAbstractOscillator.java	258
39	com/sun/media/sound/ModelByteBuffer.java	262
40	com/sun/media/sound/ModelByteBufferWavetable.java	268
41	com/sun/media/sound/ModelChannelMixer.java	273
42	com/sun/media/sound/ModelConnectionBlock.java	274
43	com/sun/media/sound/ModelDestination.java	277
44	com/sun/media/sound/ModelDirectedPlayer.java	279
45	com/sun/media/sound/ModelDirector.java	280
46	com/sun/media/sound/ModelIdentifier.java	281
47	com/sun/media/sound/ModelInstrument.java	284
48	com/sun/media/sound/ModelInstrumentComparator.java	287
49	com/sun/media/sound/ModelMappedInstrument.java	288
50	com/sun/media/sound/ModelOscillator.java	290
51	com/sun/media/sound/ModelOscillatorStream.java	291
52	com/sun/media/sound/ModelPatch.java	292
53	com/sun/media/sound/ModelPerformer.java	293
54	com/sun/media/sound/ModelSource.java	296
55	com/sun/media/sound/ModelStandardDirector.java	298

56	com/sun/media/sound/ModelStandardIndexedDirector.java	300
57	com/sun/media/sound/ModelStandardTransform.java	304
58	com/sun/media/sound/ModelTransform.java	307
59	com/sun/media/sound/ModelWavetable.java	308
60	com/sun/media/sound/RIFFInvalidDataException.java	309
61	com/sun/media/sound/RIFFInvalidFormatException.java	310
62	com/sun/media/sound/RIFFReader.java	311
63	com/sun/media/sound/RIFFWriter.java	317
64	com/sun/media/sound/RealTimeSequencer.java	323
65	com/sun/media/sound/SF2GlobalRegion.java	358
66	com/sun/media/sound/SF2Instrument.java	359
67	com/sun/media/sound/SF2InstrumentRegion.java	374
68	com/sun/media/sound/SF2Layer.java	375
69	com/sun/media/sound/SF2LayerRegion.java	377
70	com/sun/media/sound/SF2Modulator.java	378
71	com/sun/media/sound/SF2Region.java	380
72	com/sun/media/sound/SF2Sample.java	383
73	com/sun/media/sound/SF2Soundbank.java	387
74	com/sun/media/sound/SF2SoundbankReader.java	403
75	com/sun/media/sound/SimpleInstrument.java	405
76	com/sun/media/sound/SimpleSoundbank.java	409
77	com/sun/media/sound/SoftAbstractResampler.java	412
78	com/sun/media/sound/SoftAudioBuffer.java	419
79	com/sun/media/sound/SoftAudioProcessor.java	422
80	com/sun/media/sound/SoftAudioPusher.java	423
81	com/sun/media/sound/SoftChannel.java	425
82	com/sun/media/sound/SoftChannelProxy.java	451
83	com/sun/media/sound/SoftChorus.java	455

84	com/sun/media/sound/SoftControl.java	461
85	com/sun/media/sound/SoftCubicResampler.java	462
86	com/sun/media/sound/SoftEnvelopeGenerator.java	464
87	com/sun/media/sound/SoftFilter.java	469
88	com/sun/media/sound/SoftInstrument.java	479
89	com/sun/media/sound/SoftJitterCorrector.java	481
90	com/sun/media/sound/SoftLanczosResampler.java	486
91	com/sun/media/sound/SoftLimiter.java	488
92	com/sun/media/sound/SoftLinearResampler.java	492
93	com/sun/media/sound/SoftLinearResampler2.java	494
94	com/sun/media/sound/SoftLowFrequencyOscillator.java	496
95	com/sun/media/sound/SoftMainMixer.java	499
96	com/sun/media/sound/SoftMidiAudioFileReader.java	518
97	com/sun/media/sound/SoftMixingClip.java	522
98	com/sun/media/sound/SoftMixingDataLine.java	531
99	com/sun/media/sound/SoftMixingMainMixer.java	540
100	com/sun/media/sound/SoftMixingMixer.java	545
101	com/sun/media/sound/SoftMixingMixerProvider.java	554
102	com/sun/media/sound/SoftMixingSourceDataLine.java	556
103	com/sun/media/sound/SoftPerformer.java	565
104	com/sun/media/sound/SoftPointResampler.java	578
105	com/sun/media/sound/SoftProcess.java	580
106	com/sun/media/sound/SoftProvider.java	581
107	com/sun/media/sound/SoftReceiver.java	582
108	com/sun/media/sound/SoftResampler.java	584
109	com/sun/media/sound/SoftResamplerStreamer.java	585
110	com/sun/media/sound/SoftReverb.java	586
111	com/sun/media/sound/SoftShortMessage.java	595

112	com/sun/media/sound/SoftSincResampler.java	596
113	com/sun/media/sound/SoftSynthesizer.java	599
114	com/sun/media/sound/SoftTuning.java	621
115	com/sun/media/sound/SoftVoice.java	626
116	com/sun/media/sound/WaveExtensibleFileReader.java	641
117	com/sun/media/sound/WaveFloatFileReader.java	647
118	com/sun/media/sound/WaveFloatFileWriter.java	650
119	simplemidiplayer/ConfigDialog.java	653
120	simplemidiplayer/InfoFrame.java	658
121	simplemidiplayer/SimpleMidiPlayer.java	665

1 AudioRender.java

```
1 import java.io.File;
2 import java.util.HashMap;
3 import java.util.Map;
4
5 import javax.sound.midi.MidiSystem;
6 import javax.sound.midi.Receiver;
7 import javax.sound.midi.ShortMessage;
8 import javax.sound.sampled.AudioFileFormat;
9 import javax.sound.sampled.AudioFormat;
10 import javax.sound.sampled.AudioInputStream;
11 import javax.sound.sampled.AudioSystem;
12
13 import com.sun.media.sound.AudioSynthesizer;
14
15 public class AudioRender {
16
17     public static void main(String[] args) throws Exception
18     {
19         /*
20          * Open synthesizer in pull mode in the format 96000hz 24 bit stereo
21          * using Sinc interpolation for highest quality.
22          * With 1024 in max polyphony.
23          */
24         AudioFormat format = new AudioFormat(96000, 24, 2, true, false);
25         AudioSynthesizer synthesizer = (AudioSynthesizer)MidiSystem.getSynthesizer();
26         Map<String, Object> info = new HashMap<String, Object>();
27         info.put("resampleType", "sinc");
28         info.put("maxPolyphony", "1024");
29         AudioInputStream stream = synthesizer.openStream(format, info);
30
31         /*
32          * Play midi note 60 on channel 1 for 1 sec.
33          */
34         ShortMessage msg = new ShortMessage();
35         Receiver rcv = synthesizer.getReceiver();
36         msg.setMessage(ShortMessage.PROGRAM_CHANGE, 0, 48, 0);
37         rcv.send(msg, 0);
38         msg.setMessage(ShortMessage.NOTE_ON, 0, 60, 80);
39         rcv.send(msg, 0);
40         msg.setMessage(ShortMessage.NOTE_ON, 0, 60, 0);
41         rcv.send(msg, 1000000);
42
43         /*
44          * Calculate how many bytes 4 seconds are.
45          */
46         long len = (long)(format.getFrameRate() * 4);
47
48         /*
49          * Write 10 second into output file.
50          */
51         stream = new AudioInputStream(stream, format, len);
52         AudioSystem.write(stream, AudioFileFormat.Type.WAVE, new File("output.wav"));
53
54         /*
55          * Close all resources.
56          */
57         synthesizer.close();
58     }
59 }
```

2 FMTest1.java

```
1 import java.io.BufferedReader;
2 import java.io.IOException;
3 import java.io.InputStreamReader;
4
5 import javax.sound.midi.MidiSystem;
6 import javax.sound.midi.Sequence;
7 import javax.sound.midi.Sequencer;
8 import javax.sound.midi.Synthesizer;
9
10 import com.sun.media.sound.ModelAbstractOscillator;
11
12
13 public class FMTest1 {
14
15     // A implementation of very simple FM Oscillator
16     // =====
17
18     public static class MyOscillator extends ModelAbstractOscillator
19     {
20         double ix = 0;
21         double last_ix_step = -1;
22
23         public int read(float[][] buffers, int offset, int len) throws IOException {
24
25             // Grab channel 0 buffer from buffers
26             float[] buffer = buffers[0];
27
28             // Calculate ix step so sin oscillirator is tuned so 6900 cents is 440 hz
29             double target_ix_step =
30                 Math.exp((getPitch()-6900) * (Math.log(2.0) / 1200.0))
31                 * (440 / getSampleRate()) * (Math.PI*2);
32             double ix_step = last_ix_step;
33             if(ix_step == -1) ix_step = target_ix_step;
34             double ix_step_step = (target_ix_step - ix_step)/len;
35
36             // Simple FM synthesizer implementation
37             int endoffset = offset + len;
38             for (int i = offset; i < endoffset; i++) {
39                 buffer[i] = (float)Math.sin(ix + Math.sin(ix*3));
40                 ix += ix_step;
41                 // ix_step_step is used for
42                 // smooth pitch changes
43                 ix_step += ix_step_step;
44             }
45
46             last_ix_step = target_ix_step;
47
48             return len;
49         }
50
51     }
52
53
54     // This code is only for testing the instrument we defined/created above.
55     // =====
56
57     public static void main(String[] args) throws Exception {
58
59         Synthesizer synth = MidiSystem.getSynthesizer();
60         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)

```

```

185         return (short) -12000;
186     if (generator == 38)
187         return (short) -12000;
188     if (generator == 43)
189         return (short) 0x7F00;
190     if (generator == 44)
191         return (short) 0x7F00;
192     if (generator == 46)
193         return (short) -1;
194     if (generator == 47)
195         return (short) -1;
196     if (generator == 56)
197         return (short) 100;
198     if (generator == 58)
199         return (short) -1;
200     return 0;
201 }
202
203 public int getDefaultValue() {
204     return getSFDefaultValue(gid);
205 }
206
207 private short getSFMinValue(int generator) {
208     switch (generator) {
209     case 0:
210         return 0;
211     case 1:
212         return -32768;
213     case 2:
214         return -32768;
215     case 3:
216         return -32768;
217     case 4:
218         return 0;
219     case 5:
220         return -12000;
221     case 6:
222         return -12000;
223     case 7:
224         return -12000;
225     case 8:
226         return 1500;
227     case 9:
228         return 0;
229     case 10:
230         return -12000;
231     case 11:
232         return -12000;
233     case 12:
234         return -32768;
235     case 13:
236         return -960;
237     case 15:
238         return 0;
239     case 16:
240         return 0;
241     case 17:
242         return -500;
243     case 21:
244         return -16000;
245     case 22:
246         return -12000;

```

```
247     case 23:
248         return -16000;
249     case 24:
250         return -12000;
251     case 25:
252         return -12000;
253     case 26:
254         return -12000;
255     case 27:
256         return -12000;
257     case 28:
258         return -12000;
259     case 29:
260         return 0;
261     case 30:
262         return -12000;
263     case 31:
264         return -1200;
265     case 32:
266         return -1200;
267     case 33:
268         return -12000;
269     case 34:
270         return -12000;
271     case 35:
272         return -12000;
273     case 36:
274         return -12000;
275     case 37:
276         return 0;
277     case 38:
278         return -12000;
279     case 39:
280         return -1200;
281     case 40:
282         return -1200;
283     case 43:
284         return 0;
285     case 44:
286         return 0;
287     case 45:
288         return -32768;
289     case 46:
290         return 0;
291     case 47:
292         return 0;
293     case 48:
294         return 0;
295     case 50:
296         return -32768;
297     case 51:
298         return -120;
299     case 52:
300         return -99;
301     case 54:
302         return -32768;
303     case 56:
304         return 0;
305     case 57:
306         return 1;
307     case 58:
308         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:

```

```

371         return 1200;
372     case 32:
373         return 1200;
374     case 33:
375         return 5000;
376     case 34:
377         return 8000;
378     case 35:
379         return 5000;
380     case 36:
381         return 8000;
382     case 37:
383         return 1440;
384     case 38:
385         return 8000;
386     case 39:
387         return 1200;
388     case 40:
389         return 1200;
390     case 43:
391         return 127;
392     case 44:
393         return 127;
394     case 45:
395         return 32767;
396     case 46:
397         return 127;
398     case 47:
399         return 127;
400     case 48:
401         return 1440;
402     case 50:
403         return 32767;
404     case 51:
405         return 120;
406     case 52:
407         return 99;
408     case 54:
409         return 32767;
410     case 56:
411         return 1200;
412     case 57:
413         return 127;
414     case 58:
415         return 127;
416     default:
417         return Short.MAX_VALUE;
418     }
419 }
420
421 private String cent = "cent";
422
423 private String cb = "0.1dB";
424
425 private String timcent = "timcent";
426
427 private String semitone = "semitone";
428
429 private String mpercent = "0.1%";
430
431 private String negative_mpercent = "-0.1%";
432

```

```

433 private String getSFTType(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;

```



```

495     case 40:
496         return timcent;
497     case 48:
498         return cb;
499     case 51:
500         return semitone;
501     case 52:
502         return cent;
503     case 56:
504         return cent;
505     default:
506         return "";
507 }
508
509 }
510
511 public SfGeneratorEditor(final int gid) {
512     this.gid = gid;
513
514     setOpaque(false);
515     setLayout(new FlowLayout(FlowLayout.LEFT, 2, 0));
516
517     JButton defaulter = new JButton();
518     defaulter.setText("R");
519     defaulter.setMargin(new Insets(0, 0, 0, 0));
520     defaulter.addActionListener(new ActionListener() {
521         public void actionPerformed(ActionEvent arg0) {
522             slider.setValue(getSFDefaultValue(gid));
523         }
524     });
525
526     slider = new JSlider();
527     slider.setOpaque(false);
528     slider.setMinimum(getSFMinValue(gid));
529     slider.setMaximum(getSFMaxValue(gid));
530     slider.setValue(getSFDefaultValue(gid));
531     slider.setMaximumSize(new Dimension(85, 20));
532     slider.setPreferredSize(new Dimension(85, 20));
533     slider.addChangeListener(new ChangeListener() {
534         public void stateChanged(ChangeEvent arg0) {
535             dirty = true;
536             updateDisplay();
537         }
538     });
539     add(slider);
540     textfield = new JTextField("");
541     textfield.setMaximumSize(new Dimension(50, 20));
542     textfield.setPreferredSize(new Dimension(50, 20));
543     textfield.addKeyListener(new KeyListener() {
544         public void keyPressed(KeyEvent arg0) {
545             if (arg0.getKeyCode() == KeyEvent.VK_ENTER) {
546                 slider.setValue(Integer.parseInt(textfield.getText()));
547                 arg0.consume();
548             }
549         }
550
551         public void keyReleased(KeyEvent arg0) {
552         }
553
554         public void keyTyped(KeyEvent arg0) {
555         }
556     });

```

```

557     textfield.addFocusListener(new FocusListener() {
558         public void focusGained(FocusEvent arg0) {
559             }
560
561         public void focusLost(FocusEvent arg0) {
562             int x = Integer.parseInt(textfield.getText());
563             if (slider.getValue() != x)
564                 slider.setValue(x);
565             updateDisplay();
566         }
567     });
568     add(textfield);
569     updateDisplay();
570     add(defaulter);
571     add(new JLabel(getSFType(gid)));
572     generators.add(this);
573 }
574
575 public void setValue(int i) {
576     if (slider.getValue() != i)
577         slider.setValue(i);
578 }
579
580 public int getValue() {
581     return slider.getValue();
582 }
583
584 private void updateDisplay() {
585     if (textfield == null)
586         return;
587     textfield.setText("" + slider.getValue());
588 }
589
590 public void process(SF2Region region) {
591     if (slider.getValue() != getSFDefaultValue(gid)) {
592         region.putInteger(gid, slider.getValue());
593     }
594 }
595
596 }
597
598 private class SpectrumViewer extends JComponent {
599     private static final long serialVersionUID = 1L;
600
601     public void paint(Graphics g){
602         super.paint(g);
603
604         int fftlen = data.length / 25;
605
606         double base = 8 * 15;
607
608         Graphics2D g2 = (Graphics2D) g;
609         /*
610          * g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
611          * RenderingHints.VALUE_ANTIALIAS_ON);
612          * g2.setRenderingHint(RenderingHints.KEY_FRACTIONALMETRICS,
613          * RenderingHints.VALUE_FRACTIONALMETRICS_ON);
614          */
615         int w = getWidth();
616         int h = getHeight() - 20;
617
618         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 private Receiver recv;
682
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 SpectrumViewer 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

```

```

743 FFT ifft_obj = null;
744
745 int ifft_obj_len = 0;
746
747 public void ifft(double[] data) {
748     if (ifft_obj == null || ifft_obj_len != data.length / 2) {
749         ifft_obj_len = data.length / 2;
750         ifft_obj = new FFT(ifft_obj_len, 1);
751     }
752     ifft_obj.transform(data);
753 }
754
755 public SF2Sample newSimpleFFTSample(SF2Soundbank sf2, String name,
756     double[] data, double base) {
757     return newSimpleFFTSample(sf2, name, data, base, 10);
758 }
759
760 public float[] toFloat(double[] in) {
761     float[] out = new float[in.length];
762     for (int i = 0; i < out.length; i++) {
763         out[i] = (float) in[i];
764     }
765     return out;
766 }
767
768 public float[] loopExtend(float[] data, int newsize) {
769     float[] outdata = new float[newsize];
770     int p_len = data.length;
771     int p_ps = 0;
772     for (int i = 0; i < outdata.length; i++) {
773         outdata[i] = data[p_ps];
774         p_ps++;
775         if (p_ps == p_len)
776             p_ps = 0;
777     }
778     return outdata;
779 }
780
781 public void normalize(double[] data, double target) {
782     double maxvalue = 0;
783     for (int i = 0; i < data.length; i++) {
784         if (data[i] > maxvalue)
785             maxvalue = data[i];
786         if (-data[i] > maxvalue)
787             maxvalue = -data[i];
788     }
789     if (maxvalue == 0)
790         return;
791     double gain = target / maxvalue;
792     for (int i = 0; i < data.length; i++)
793         data[i] *= gain;
794 }
795
796 public void fadeUp(float[] data, int samples) {
797     double dsamples = samples;
798     for (int i = 0; i < samples; i++)
799         data[i] *= i / dsamples;
800 }
801
802 public byte[] toBytes(float[] in, AudioFormat format) {
803     byte[] out = new byte[in.length * format.getFrameSize()];
804 }

```

```

805         return AudioFloatConverter.getConverter(format).toByteArray(in, out);
806     }
807
808     public SF2Sample newSimpleFFTSample(SF2Soundbank sf2, String name,
809         double[] data, double base, int fadeuptime) {
810
811         int fftsize = data.length / 2;
812         AudioFormat format = new AudioFormat(44100, 16, 1, true, false);
813         double basefreq = (base / fftsize) * format.getSampleRate() * 0.5;
814
815         randomPhase(data);
816         ifft(data);
817         data = realPart(data);
818         normalize(data, 0.9);
819         float[] fdata = toFloat(data);
820         fdata = loopExtend(fdata, fdata.length + 512);
821         fadeUp(fdata, fadeuptime);
822         byte[] bdata = toBytes(fdata, format);
823
824         /*
825          * Create SoundFont2 sample.
826          */
827         SF2Sample sample = new SF2Sample(sf2);
828         sample.setName(name);
829         sample.setData(bdata);
830         sample.setStartLoop(256);
831         sample.setEndLoop(fftsize + 256);
832         sample.setSampleRate((long) format.getSampleRate());
833         double orgnote = (69 + 12)
834             + (12 * Math.log(basefreq / 440.0) / Math.log(2));
835         sample.setOriginalPitch((int) orgnote);
836         sample.setPitchCorrection((byte) (-(orgnote - (int) orgnote) * 100.0));
837         sf2.addResource(sample);
838
839         return sample;
840     }
841
842     public Soundbank designInstrument(boolean createsbk) {
843
844         int x = 8;
845         int fftsize = 4096 * x;
846         if (data == null || data.length != fftsize * 2)
847             data = new double[fftsize * 2];
848
849         Arrays.fill(data, 0);
850
851         double[] data = this.data;
852         double base = x * 15;
853
854         for (int i = 0; i < width_sliders.length; i++) {
855             double h = i + 1;
856             try {
857                 h = Double.parseDouble(harmonic_field[i].getText());
858             } catch (NumberFormatException e) {
859                 harmonic_field[i].setText("" + (i + 1));
860             }
861             if (gain_sliders[i].getValue() > 0) {
862                 double width = (width_sliders[i].getValue()) / 4;
863                 double db = -(100 - gain_sliders[i].getValue());
864                 double gain = Math.pow(10, db / 20.0);
865                 complexGaussianDist(data, base * h, width, gain);
866             }
867         }
868     }

```

```

867     }
868
869     if (!createsbk)
870         return null;
871
872     SF2Soundbank sf2 = new SF2Soundbank();
873     sf2.setName("My_SoundFont");
874     sf2.setVendor("Generated");
875     sf2.setDescription("A_newly_created_soundfont");
876
877     SF2Layer layer1 = null;
878     SF2Layer layer2 = null;
879
880     for (int i = 0; i < 2; i++) {
881
882         double[] data_audio = i == 0 ? this.data_audio : this.data_audio2;
883         if (data_audio == null || data_audio.length != fftsize * 2) {
884             data_audio = new double[fftsize * 2];
885             if (i == 0)
886                 this.data_audio = data_audio;
887             else
888                 this.data_audio2 = data_audio;
889         }
890         System.arraycopy(data, 0, data_audio, 0, data_audio.length);
891
892         SF2Sample sample = newSimpleFFTSample(sf2, "FFT_Sample_" + i,
893             data_audio, base, 200);
894         SF2Layer layer = newLayer(sf2, "FFT_Layer_" + i, sample);
895         SF2Region region = layer.getRegions().get(0);
896         region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1); // set loop
897         if (stereo_mode)
898             if (i == 0)
899                 region.putInteger(SF2Region.GENERATOR_PAN, -500);
900             else
901                 region.putInteger(SF2Region.GENERATOR_PAN, 500);
902
903         for (SfGeneratorEditor gen : generators) {
904             gen.process(region);
905         }
906
907         if (i == 0)
908             layer1 = layer;
909         else
910             layer2 = layer;
911
912         if (!stereo_mode)
913             break;
914     }
915
916     if (!stereo_mode)
917         newInstrument(sf2, "FFT_Instrument", new Patch(0, 0), layer1);
918     else
919         newInstrument(sf2, "FFT_Instrument", new Patch(0, 0), layer1,
920             layer2);
921
922     return sf2;
923
924 }
925
926 public SF2Layer newLayer(SF2Soundbank sf2, String name, SF2Sample sample) {
927     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();

```



```

991         SwingUtilities.invokeLater(new Runnable() {
992             public void run() {
993                 createGUI();
994                 validate();
995                 invalidate();
996             }
997         });
998     });
999
1000     } catch (Exception e) {
1001         e.printStackTrace();
1002     }
1003 }
1004 };
1005 new Thread(runnable).start();
1006 }
1007
1008 public void createGUI() {
1009     remove(firstpanel);
1010     JPanel toppanel = new JPanel();
1011     toppanel.setBackground(Color.WHITE);
1012     toppanel.setLayout(new FlowLayout());
1013     add(toppanel);
1014
1015     JPanel boxpanel = new JPanel();
1016     boxpanel.setOpaque(false);
1017     boxpanel.setLayout(new BoxLayout(boxpanel, BoxLayout.Y_AXIS));
1018     toppanel.add(boxpanel);
1019
1020     Dimension vdim12 = new Dimension(1000, 50);
1021     VirtualKeyboard12 vkeyboard12 = new VirtualKeyboard12();
1022     vkeyboard12.setSize(vdim12);
1023     vkeyboard12.setPreferredSize(vdim12);
1024     vkeyboard12.setMinimumSize(vdim12);
1025     vkeyboard12.setMaximumSize(vdim12);
1026     vkeyboard12.setChannel(0);
1027
1028     final MidiChannel channel1 = synth.getChannels()[0];
1029     channel1.controlChange(7, 127);
1030
1031     Receiver recv2 = new Receiver() {
1032         public void close() {
1033         }
1034
1035         public void send(MidiMessage arg0, long arg1) {
1036             if (dirty) {
1037                 synth.unloadAllInstruments(sbk);
1038                 sbk = designInstrument(true);
1039                 synth.loadAllInstruments(sbk);
1040                 channel1.programChange(0);
1041                 dirty = false;
1042             }
1043             recv.send(arg0, arg1);
1044         }
1045     };
1046
1047     vkeyboard12.setReceiver(recv2);
1048
1049     JScrollPane scrollpane12 = new JScrollPane(vkeyboard12);
1050     scrollpane12.setPreferredSize(new Dimension(500, 80));
1051     scrollpane12.getViewPort().setViewPosition(new Point(200, 0));

```

```

1053 JPanel panel12 = new JPanel(new BorderLayout());
1054 panel12.setOpaque(false);
1055 panel12.add(scrollpane12);
1056
1057 JPanel panelslides = new JPanel(new GridBagLayout());
1058
1059 GridBagConstraints c = new GridBagConstraints();
1060 c.gridy = 0;
1061 c.gridx = 0;
1062 panelslides.add(new JLabel("Harmonic"), c);
1063 c.gridx = 1;
1064 panelslides.add(new JLabel("Amplitude"), c);
1065 c.gridx = 2;
1066 panelslides.add(new JLabel("Bandwidth"), c);
1067
1068 final ChangeListener changelistener = new ChangeListener() {
1069     public void stateChanged(ChangeEvent arg0) {
1070         if (haltScreenUpdates)
1071             return;
1072         dirty = true;
1073         if (sv != null) {
1074             designInstrument(false);
1075             sv.repaint();
1076         }
1077     }
1078 };
1079
1080 for (int i = 0; i < width_sliders.length; i++) {
1081
1082     JSlider amp_slider = new JSlider();
1083     amp_slider.setOpaque(false);
1084     amp_slider.setMinimum(0);
1085     amp_slider.setMaximum(100);
1086     amp_slider.setValue(0);
1087     amp_slider.addChangeListener(changelistener);
1088
1089     JSlider w_slider = new JSlider();
1090     w_slider.setOpaque(false);
1091     w_slider.setMinimum(0);
1092     w_slider.setMaximum(100);
1093     w_slider.setValue(0);
1094     w_slider.addChangeListener(changelistener);
1095
1096     JTextField tf = new JTextField("" + (i + 1));
1097     tf.setMaximumSize(new Dimension(50, 20));
1098     tf.setPreferredSize(new Dimension(50, 20));
1099     tf.addKeyListener(new KeyListener() {
1100         public void keyPressed(KeyEvent arg0) {
1101             if (arg0.getKeyCode() == KeyEvent.VK_ENTER) {
1102                 changelistener.stateChanged(null);
1103             }
1104         }
1105
1106         public void keyReleased(KeyEvent arg0) {
1107         }
1108
1109         public void keyTyped(KeyEvent arg0) {
1110         }
1111     });
1112     tf.addFocusListener(new FocusListener() {
1113         public void focusGained(FocusEvent arg0) {

```

```

1115     }
1116
1117     public void focusLost(FocusEvent arg0) {
1118         changelistener.stateChanged(null);
1119     }
1120 });
1121
1122     harmonic_field[i] = tf;
1123     width_sliders[i] = w_slider;
1124     gain_sliders[i] = amp_slider;
1125
1126     c.gridy = i + 1;
1127     c.gridx = 0;
1128     // panelslides.add(new JLabel("" + (i + 1)), c);
1129     panelslides.add(tf, c);
1130     c.gridx = 1;
1131     panelslides.add(amp_slider, c);
1132     c.gridx = 2;
1133     panelslides.add(w_slider, c);
1134
1135 }
1136 gain_sliders[0].setValue(100);
1137 GridBagConstraints ec = new GridBagConstraints();
1138 ec.gridy = 100;
1139 ec.gridwidth = 2;
1140 ec.weightx = 10;
1141 ec.weighty = 10;
1142
1143 c.anchor = GridBagConstraints.LINE_START;
1144 JPanel volenv = new JPanel(new GridBagLayout());
1145 volenv.setBorder(BorderFactory.createTitledBorder("Volume_Envelope"));
1146 volenv.setOpaque(false);
1147 c.gridy = 0;
1148 c.gridx = 0;
1149 volenv.add(new JLabel("Delay"), c);
1150 c.gridx = 1;
1151 volenv.add(new SfGeneratorEditor(SF2Region.GENERATOR_DELAYVOLENV), c);
1152 c.gridy = 1;
1153 c.gridx = 0;
1154 volenv.add(new JLabel("Attack"), c);
1155 c.gridx = 1;
1156 volenv.add(new SfGeneratorEditor(SF2Region.GENERATOR_ATTACKVOLENV), c);
1157 c.gridy = 2;
1158 c.gridx = 0;
1159 volenv.add(new JLabel("Hold"), c);
1160 c.gridx = 1;
1161 volenv.add(new SfGeneratorEditor(SF2Region.GENERATOR_HOLDVOLENV), c);
1162 c.gridy = 3;
1163 c.gridx = 0;
1164 volenv.add(new JLabel("Decay"), c);
1165 c.gridx = 1;
1166 volenv.add(new SfGeneratorEditor(SF2Region.GENERATOR_DECAYVOLENV), c);
1167 c.gridy = 4;
1168 c.gridx = 0;
1169 volenv.add(new JLabel("Sustain"), c);
1170 c.gridx = 1;
1171 volenv.add(new SfGeneratorEditor(SF2Region.GENERATOR_SUSTAINVOLENV), c);
1172 c.gridy = 5;
1173 c.gridx = 0;
1174 volenv.add(new JLabel("Release"), c);
1175 c.gridx = 1;
1176 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();

```

```

1301 c2.fill = GridBagConstraints.BOTH;
1302 JPanel generators = new JPanel(new GridBagLayout());
1303 generators.setBorder(BorderFactory.createEmptyBorder(2, 2, 2, 2));
1304 generators.setOpaque(false);
1305 c2.gridx = 0;
1306 c2.gridy = 0;
1307 generators.add(volenv, c2);
1308 c2.gridx = 1;
1309 c2.gridy = 0;
1310 generators.add(modenv, c2);
1311 c2.gridx = 0;
1312 c2.gridy = 1;
1313 generators.add(lfo_vib, c2);
1314 c2.gridx = 1;
1315 c2.gridy = 1;
1316 generators.add(lfo_mod, c2);
1317 c2.gridx = 0;
1318 c2.gridy = 2;
1319 generators.add(filter_gen, c2);
1320 generators.add(new JLabel(), ec);
1321
1322 sbk = designInstrument(true);
1323
1324 synth.unloadAllInstruments(synth.getDefaultSoundbank());
1325 synth.loadAllInstruments(sbk);
1326 channel1.programChange(0);
1327
1328 panelslides.setOpaque(false);
1329 panelslides.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
1330
1331 sv = new SpecturmViewer();
1332 Dimension dim = new Dimension(300, 200);
1333 sv.setMinimumSize(dim);
1334 sv.setPreferredSize(dim);
1335
1336 initPresets();
1337 final JComboBox presetSelector = new JComboBox(presets.toArray());
1338 presetSelector.addActionListener(new ActionListener() {
1339
1340     public void actionPerformed(ActionEvent arg0) {
1341         ((Preset) presetSelector.getSelectedItem()).select();
1342     }
1343
1344 });
1345
1346 JPanel instrumentpanel = new JPanel();
1347 instrumentpanel.setLayout(new FlowLayout(FlowLayout.LEFT));
1348 instrumentpanel.setOpaque(false);
1349 instrumentpanel.add(new JLabel("Preset:"));
1350 instrumentpanel.add(presetSelector);
1351 final JCheckBox reverbcheckbox = new JCheckBox("Reverb");
1352 reverbcheckbox.setOpaque(false);
1353 reverbcheckbox.addActionListener(new ActionListener() {
1354     public void actionPerformed(ActionEvent arg0) {
1355         channel1.controlChange(91, reverbcheckbox.isSelected() ? 127
1356             : 0);
1357     }
1358 });
1359 channel1.controlChange(91, 0);
1360 instrumentpanel.add(reverbcheckbox);
1361 final JCheckBox choruscheckbox = new JCheckBox("Chorus");
1362 choruscheckbox.setOpaque(false);

```

```

1363 choruscheckbox.addActionListener(new ActionListener() {
1364     public void actionPerformed(ActionEvent arg0) {
1365         channel1.controlChange(93, choruscheckbox.isSelected() ? 127
1366             : 0);
1367     }
1368 });
1369 channel1.controlChange(93, 0);
1370 instrumentpanel.add(choruscheckbox);
1371 final JCheckBox portamentocheckbox = new JCheckBox("Portamento");
1372 portamentocheckbox.setOpaque(false);
1373 portamentocheckbox.addActionListener(new ActionListener() {
1374     public void actionPerformed(ActionEvent arg0) {
1375
1376         if (portamentocheckbox.isSelected()) {
1377             channel1.controlChange(126, 1); // Mono Mode
1378             channel1.controlChange(65, 127); // Set Portamento On
1379             channel1.controlChange(5, 67); // Set portamento time
1380         } else {
1381             channel1.controlChange(127, 1); // Poly Mode
1382             channel1.controlChange(65, 0); // Set Portamento off
1383         }
1384     }
1385 });
1386 instrumentpanel.add(portamentocheckbox);
1387
1388 final JCheckBox stereo_mode_checkbox = new JCheckBox("Stereo");
1389 stereo_mode_checkbox.setOpaque(false);
1390 stereo_mode_checkbox.addActionListener(new ActionListener() {
1391     public void actionPerformed(ActionEvent arg0) {
1392         changelister.stateChanged(null);
1393         stereo_mode = stereo_mode_checkbox.isSelected();
1394     }
1395 });
1396 instrumentpanel.add(stereo_mode_checkbox);
1397
1398 boxpanel.add(instrumentpanel);
1399 boxpanel.add(panel12);
1400 boxpanel.add(sv);
1401
1402 JPanel presets_panel = new JPanel();
1403 presets_panel.setLayout(new BoxLayout(presets_panel, BoxLayout.Y_AXIS));
1404 presets_panel.setOpaque(false);
1405
1406 JTabbedPane tabs = new JTabbedPane();
1407 tabs.addTab("Harmonic_Profile", panelslides);
1408 tabs.addTab("Instrument_Settings", generators);
1409 boxpanel.add(tabs);
1410
1411 }
1412
1413 public void initPresets() {
1414     Preset preset;
1415
1416     preset = new Preset("Sine_wave");
1417     presets.add(preset);
1418
1419     preset = new Preset("Square_wave");
1420     for (int i = 1; i <= 20; i += 2) {
1421         double x = 1.0 / (double) i;
1422         preset.setHarmonic(i, (int) (20 * Math.log10(x)), 0);
1423     }
1424     presets.add(preset);

```

```

1425 preset = new Preset("Triangle_wave");
1426 for (int i = 1; i <= 20; i += 2) {
1427     double x = 1.0 / (double) (i * i);
1428     preset.setHarmonic(i, (int) (20 * Math.log10(x)), 0);
1429 }
1430 presets.add(preset);
1431 preset = new Preset("Sawtooth_wave");
1432 for (int i = 1; i <= 20; i++) {
1433     double x = 1.0 / (double) i;
1434     preset.setHarmonic(i, (int) (20 * Math.log10(x)), 0);
1435 }
1436 presets.add(preset);
1437
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
1462 preset = new Preset("Flute");
1463 preset.setHarmonic(1, 0, 0);
1464 preset.setHarmonic(2, 0, 0);
1465 for (int i = 2; i < 20; i++) {
1466     preset.setHarmonic(i + 1, -(i-2) * 5 - 10, 0);
1467 }
1468 preset.setGenerator(SF2Region.GENERATOR_ATTACKVOLENV, -6000);
1469 preset.setGenerator(SF2Region.GENERATOR_RELEASEVOLENV, -1000);
1470 preset.setGenerator(SF2Region.GENERATOR_DECAYVOLENV, 4000);
1471 preset.setGenerator(SF2Region.GENERATOR_SUSTAINVOLENV, -100);
1472 preset.setGenerator(SF2Region.GENERATOR_INITIALFILTERFC, 9500);
1473 presets.add(preset);
1474
1475
1476 preset = new Preset("Flute_2");
1477 preset.setHarmonic(1, 0, 0);
1478 preset.setHarmonic(2, -25, 0);
1479 preset.setHarmonic(3, -15, 0);
1480 preset.setHarmonic(4, -30, 0);
1481 preset.setHarmonic(5, -25, 0);
1482 preset.setHarmonic(6, -40, 0);
1483 preset.setHarmonic(7, -35, 0);
1484 preset.setHarmonic(8, -45, 0);
1485 preset.setHarmonic(9, -55, 0);
1486 preset.setHarmonic(10, -60, 0);
1487 preset.setHarmonic(11, -65, 0);

```



```

1487 preset.setHarmonic(12, -55, 0);
1488 preset.setHarmonic(13, -65, 0);
1489 preset.setHarmonic(14, -70, 0);
1490 preset.setGenerator(SF2Region.GENERATOR_ATTACKVOLENV, -6000);
1491 preset.setGenerator(SF2Region.GENERATOR_RELEASEVOLENV, -1000);
1492 preset.setGenerator(SF2Region.GENERATOR_DECAYVOLENV, 4000);
1493 preset.setGenerator(SF2Region.GENERATOR_SUSTAINVOLENV, -100);
1494 preset.setGenerator(SF2Region.GENERATOR_INITIALFILTERFC, 9500);
1495 presets.add(preset);
1496
1497 preset = new Preset("Trumpet");
1498 preset.setHarmonic(1, -20, 0);
1499 preset.setHarmonic(2, -15, 0);
1500 preset.setHarmonic(3, -8, 0);
1501 for (int i = 3; i < 20; i++) {
1502     preset.setHarmonic(i + 1, -i * 4, 0);
1503 }
1504 preset.setGenerator(SF2Region.GENERATOR_ATTACKVOLENV, -10000);
1505 preset.setGenerator(SF2Region.GENERATOR_RELEASEVOLENV, 0);
1506 preset.setGenerator(SF2Region.GENERATOR_DECAYVOLENV, 4000);
1507 preset.setGenerator(SF2Region.GENERATOR_SUSTAINVOLENV, -100);
1508
1509 preset.setGenerator(SF2Region.GENERATOR_ATTACKMODENV, -4000);
1510 preset.setGenerator(SF2Region.GENERATOR_RELEASEMODENV, -2500);
1511 preset.setGenerator(SF2Region.GENERATOR_MODENVTOFILTERFC, 5000);
1512 preset.setGenerator(SF2Region.GENERATOR_INITIALFILTERFC, 4500);
1513 preset.setGenerator(SF2Region.GENERATOR_INITIALFILTERQ, 10);
1514
1515 presets.add(preset);
1516
1517 preset = new Preset("Horn");
1518 preset.setHarmonic(1, -10, 0);
1519 for (int i = 1; i < 20; i++) {
1520     preset.setHarmonic(i + 1, -i * 3, 0);
1521 }
1522 preset.setGenerator(SF2Region.GENERATOR_SAMPLEMODES, 1);
1523 preset.setGenerator(SF2Region.GENERATOR_ATTACKVOLENV, -6000);
1524 preset.setGenerator(SF2Region.GENERATOR_RELEASEVOLENV, -1000);
1525 preset.setGenerator(SF2Region.GENERATOR_DECAYVOLENV, 4000);
1526 preset.setGenerator(SF2Region.GENERATOR_SUSTAINVOLENV, -100);
1527
1528 preset.setGenerator(SF2Region.GENERATOR_ATTACKMODENV, -500);
1529 preset.setGenerator(SF2Region.GENERATOR_RELEASEMODENV, 12000);
1530 preset.setGenerator(SF2Region.GENERATOR_MODENVTOFILTERFC, 5000);
1531 preset.setGenerator(SF2Region.GENERATOR_INITIALFILTERFC, 4500);
1532
1533 presets.add(preset);
1534
1535 preset = new Preset("Strings");
1536 for (int i = 0; i < 20; i++) {
1537     preset.setHarmonic(i + 1, -i * 4, i * 4 + 5);
1538 }
1539 preset.setGenerator(SF2Region.GENERATOR_ATTACKVOLENV, -4000);
1540 preset.setGenerator(SF2Region.GENERATOR_RELEASEVOLENV, 2000);
1541
1542 presets.add(preset);
1543
1544 preset = new Preset("Choir");
1545 for (int i = 0; i < 20; i++) {
1546     preset.setHarmonic(i + 1, -i * 4, i * 4 + 10);
1547 }
1548 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 *
1572             5,
1573             i * 1 + 5);
1574     }
1575     preset.setGenerator(SF2Region.GENERATOR_SUSTAINVOLENV, 1440);
1576     preset.setGenerator(SF2Region.GENERATOR_RELEASEVOLENV, 4000);
1577     preset.setGenerator(SF2Region.GENERATOR_DECAYVOLENV, 4000);
1578     preset.setGenerator(SF2Region.GENERATOR_INITIALFILTERFC, 9500);
1579
1580     presets.add(preset);
1581
1582     preset = new Preset("Musical_Bell");
1583     for (int i = 0; i < 20; i++) {
1584         preset.setHarmonic(i + 1, (i + 1) + (i == 0 ? 0 : 0.2), (i == 0 ? 0
1585             : -35)
1586             + (i == 16 ? 30 : 0)
1587             + (int) (20 * Math.log10(1.0 / (i * i + 1.0))),
1588             (i == 16 ? 20 : 0) + i);
1589     }
1590     preset.setGenerator(SF2Region.GENERATOR_SUSTAINVOLENV, 1440);
1591     preset.setGenerator(SF2Region.GENERATOR_RELEASEVOLENV, 4000);
1592     preset.setGenerator(SF2Region.GENERATOR_DECAYVOLENV, 4000);
1593
1594     presets.add(preset);
1595
1596     preset = new Preset("New_Age");
1597     for (int i = 0; i < 20; i++) {
1598         preset.setHarmonic(i + 1, -i * 4, i * 4 + 5);
1599     }
1600     preset.setHarmonic(20, -30, 85);
1601     preset.setGenerator(SF2Region.GENERATOR_ATTACKVOLENV, -4000);
1602     preset.setGenerator(SF2Region.GENERATOR_RELEASEVOLENV, 4000);
1603     preset.setGenerator(SF2Region.GENERATOR_DECAYVOLENV, -1200);
1604     preset.setGenerator(SF2Region.GENERATOR_SUSTAINVOLENV, 150);
1605
1606     presets.add(preset);
1607 }
1608
1609 }

```

4 MakeSoundFont.java

```
1  /*
2   * Copyright (c) 2007 by Karl Helgason
3   * All rights reserved.
4   *
5   * Redistribution and use in source and binary forms, with or without
6   * modification, are permitted provided that the following conditions
7   * are met:
8   *
9   * - Redistributions of source code must retain the above copyright notice,
10  *   this list of conditions and the following disclaimer.
11  * - Redistributions in binary form must reproduce the above copyright
12  *   notice, this list of conditions and the following disclaimer in the
13  *   documentation and/or other materials provided with the distribution.
14  *
15  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
16  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
17  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
18  * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
19  * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
20  * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
21  * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
22  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
23  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
24  * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
25  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
26  * OF THE POSSIBILITY OF SUCH DAMAGE.
27  */
28
29  import java.io.File;
30  import java.io.IOException;
31
32  import javax.sound.sampled.*;
33
34  import com.sun.media.sound.*;
35
36  public class MakeSoundFont {
37
38      public static void main(String[] args)
39          throws UnsupportedOperationException, IOException {
40          /*
41           * Create new SoundFont2 soundbank
42           */
43          SF2Soundbank sf2 = new SF2Soundbank();
44
45          /*
46           * Select audio file.
47           */
48          File audiofile = new File("ding.wav");
49          AudioInputStream audiosream = AudioSystem
50              .getAudioInputStream(audiofile);
51
52          /*
53           * Make sure the audio stream is in
54           * correct format for soundfonts
55           * e.g.16 bit signed, little endian
56           */
57          AudioFormat format = new AudioFormat(audiosream.getFormat()
58              .getSampleRate(), 16, 1, true, false);
59          AudioInputStream convaudiosream = AudioSystem.getAudioInputStream(
60              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

```
1  /*
2   * Copyright (c) 2007 by Karl Helgason
3   * All rights reserved.
4   *
5   * Redistribution and use in source and binary forms, with or without
6   * modification, are permitted provided that the following conditions
7   * are met:
8   *
9   * - Redistributions of source code must retain the above copyright notice,
10  *   this list of conditions and the following disclaimer.
11  * - Redistributions in binary form must reproduce the above copyright
12  *   notice, this list of conditions and the following disclaimer in the
13  *   documentation and/or other materials provided with the distribution.
14  *
15  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
16  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
17  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
18  * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
19  * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
20  * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
21  * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
22  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
23  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
24  * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
25  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
26  * OF THE POSSIBILITY OF SUCH DAMAGE.
27  */
28
29  import java.io.File;
30  import java.io.FileNotFoundException;
31
32  import javax.sound.midi.MetaMessage;
33  import javax.sound.midi.MidiDevice;
34  import javax.sound.midi.MidiEvent;
35  import javax.sound.midi.MidiMessage;
36  import javax.sound.midi.MidiSystem;
37  import javax.sound.midi.MidiUnavailableException;
38  import javax.sound.midi.Receiver;
39  import javax.sound.midi.Sequence;
40  import javax.sound.midi.Soundbank;
41  import javax.sound.midi.Synthesizer;
42  import javax.sound.midi.Track;
43  import javax.sound.midi.MidiDevice.Info;
44  import javax.sound.sampled.AudioFileFormat;
45  import javax.sound.sampled.AudioInputStream;
46  import javax.sound.sampled.AudioSystem;
47
48  import com.sun.media.sound.AudioSynthesizer;
49
50  public class Midi2WavRender {
51
52      public static void main(String[] args) {
53          if (args.length >= 2)
54              try {
55                  File midi_file = new File(args[0]);
56                  if (!midi_file.exists())
57                      throw new FileNotFoundException();
58                  Sequence sequence = MidiSystem.getSequence(midi_file);
59                  Soundbank soundbank = null;
60                  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_AudioSynthesizer_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);
```

```

61
62         infolabel = new JLabel(
63             "Loading_synthesizer,_please_wait_...");
64
65         firstpanel.add(infolabel);
66
67         validate();
68         invalidate();
69     }
70 });
71
72     synth.getDefaultSoundbank();
73     synth.open();
74     recv = synth.getReceiver();
75
76     SwingUtilities.invokeLater(new Runnable() {
77         public void run() {
78             createGUI();
79             validate();
80             invalidate();
81         }
82     });
83
84     } catch (Exception e) {
85         e.printStackTrace();
86     }
87 }
88 };
89 new Thread(runnable).start();
90 }
91
92 public JSlider createMidiControlSlider(final int control, int defaultvalue)
93 {
94     final JSlider slider = new JSlider(JSlider.HORIZONTAL);
95     slider.setMaximum(127);
96     slider.setMinimum(0);
97     slider.setValue(defaultvalue);
98     slider.addChangeListener(new ChangeListener()
99     {
100         public void stateChanged(ChangeEvent e) {
101             try {
102                 ShortMessage sms = new ShortMessage();
103                 sms.setMessage(ShortMessage.CONTROL_CHANGE, control, slider.getValue());
104                 recv.send(sms, -1);
105             } catch (InvalidMidiDataException e1) {
106                 e1.printStackTrace();
107             }
108         }
109     });
110     slider.setOpaque(false);
111     return slider;
112 }
113
114
115 public JComboBox createReverbTypeCombobox()
116 {
117     String[] reverbtypes = {"Small_Room", "Medium_Room", "Large_Room", "Medium_Hall", "Large_
        Hall", "Plate"};
118     final JComboBox combobox = new JComboBox(reverbtypes);
119     combobox.setSelectedIndex(4);
120     combobox.addActionListener(new ActionListener()
121     {

```

```

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     slider.setOpaque(false);
165     return slider;
166 }
167
168 public JComboBox createChorusTypeCombobox()
169 {
170     String[] chorustypes = {"Chorus_1", "Chorus_2", "Chorus_3", "Chorus_4", "FB_Chorus", "
        Flanger"};
171     final JComboBox combobox = new JComboBox(chorustypes);
172     combobox.setSelectedIndex(2);
173     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     slider.setOpaque(false);
219     return slider;
220 }
221
222 public JSlider createChorusModDepthSlider()
223 {
224     final JSlider slider = new JSlider(JSlider.HORIZONTAL);
225     slider.setMaximum(127);
226     slider.setMinimum(0);
227     slider.setValue(64);
228 }

```

```

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 slider.setOpaque(false);
244 return slider;
245 }
246
247 public JSlider createChorusModRateSlider()
248 {
249     final JSlider slider = new JSlider(JSlider.HORIZONTAL);
250     slider.setMaximum(127);
251     slider.setMinimum(0);
252     slider.setValue(64);
253     slider.addChangeListener(new ChangeListener()
254     {
255         public void stateChanged(ChangeEvent e) {
256             try {
257                 MidiMessage msg = UniversalSysExBuilder.DeviceControl.ChorusEffect.
                setChorusModRate(
258                 UniversalSysExBuilder.ALL_DEVICES, slider.getValue());
259                 recv.send(msg, -1);
260             } catch (IOException e1) {
261                 e1.printStackTrace();
262             } catch (InvalidMidiDataException e1) {
263                 e1.printStackTrace();
264             }
265         }
266     });
267     slider.setOpaque(false);
268     return slider;
269 }
270
271 public JSlider createChorusSendToReverbSlider()
272 {
273     final JSlider slider = new JSlider(JSlider.HORIZONTAL);
274     slider.setMaximum(127);
275     slider.setMinimum(0);
276     slider.setValue(64);
277     slider.addChangeListener(new ChangeListener()
278     {
279         public void stateChanged(ChangeEvent e) {
280             try {
281                 MidiMessage msg = UniversalSysExBuilder.DeviceControl.ChorusEffect.
                setChorusSendToReverb(
282                 UniversalSysExBuilder.ALL_DEVICES, slider.getValue());
283                 recv.send(msg, -1);
284             } catch (IOException e1) {
285                 e1.printStackTrace();
286             }
287         }
288     });
289     slider.setOpaque(false);
290     return slider;
291 }

```

```

288         } catch (InvalidMidiDataException e1) {
289             e1.printStackTrace();
290         }
291     }
292 });
293 slider.setOpaque(false);
294 return slider;
295 }
296
297 public void createGUI() {
298     remove(firstpanel);
299     JPanel toppanel = new JPanel();
300     toppanel.setBackground(Color.WHITE);
301     toppanel.setLayout(new FlowLayout());
302     add(toppanel);
303
304     JPanel boxpanel = new JPanel();
305     boxpanel.setOpaque(false);
306     boxpanel.setLayout(new BoxLayout(boxpanel, BoxLayout.Y_AXIS));
307     toppanel.add(boxpanel);
308
309     Soundbank sbk = synth.getDefaultSoundbank();
310     String[] instruments = new String[128];
311     for (int i = 0; i < instruments.length; i++) {
312         instruments[i] = i + " " + sbk.getInstrument(new Patch(0, i)).getName();
313     }
314     final JComboBox instrumentcombobox = new JComboBox(instruments);
315     instrumentcombobox.addActionListener(new ActionListener()
316     {
317         public void actionPerformed(ActionEvent e) {
318             int pgm = instrumentcombobox.getSelectedIndex();
319             synth.getChannels()[0].programChange(pgm);
320         }
321     });
322
323
324     JPanel instrumentpanel = new JPanel();
325     instrumentpanel.setLayout(new FlowLayout(FlowLayout.LEFT));
326     instrumentpanel.setOpaque(false);
327     instrumentpanel.add(new JLabel("Instrument: "));
328     instrumentpanel.add(instrumentcombobox);
329     boxpanel.add(instrumentpanel);
330
331     JPanel effectscontainer = new JPanel();
332     effectscontainer.setLayout(new BoxLayout(effectscontainer, BoxLayout.X_AXIS));
333     effectscontainer.setOpaque(false);
334     boxpanel.add(effectscontainer);
335
336     GridBagConstraints c = new GridBagConstraints();
337     c.anchor = GridBagConstraints.WEST;
338     c.insets = new Insets(2,2,2,2);
339
340     JPanel reverbeffect = new JPanel();
341     reverbeffect.setBorder(BorderFactory
342         .createTitledBorder("Reverb"));
343     reverbeffect.setOpaque(false);
344     reverbeffect.setLayout(new GridBagLayout());
345
346     c.gridx = 0;
347     c.gridy = 0;
348     reverbeffect.add(new JLabel("Level"),c);
349

```

```

350     c.gridx = 1;
351     reverbeffect.add(createMidiControlSlider(91, 40),c);
352
353     c.gridx = 0;
354     c.gridy = 1;
355     reverbeffect.add(new JLabel("Type"),c);
356     c.gridx = 1;
357     c.weightx = 1;
358     reverbeffect.add(createReverbTypeCombobox(),c);
359
360     c.gridx = 0;
361     c.gridy = 2;
362     reverbeffect.add(new JLabel("Room_size"),c);
363     c.gridx = 1;
364     reverbeffect.add(createReverbTimeSlider(),c);
365
366     c.gridx = 0;
367     c.gridy = 3;
368     c.weighty = 1;
369     JPanel emptypanel = new JPanel();
370     emptypanel.setOpaque(false);
371     reverbeffect.add(emptypanel, c);
372     c.weighty = 0;
373
374     effectscontainer.add(reverbeffect);
375
376     JPanel choruseffect = new JPanel();
377     choruseffect.setBorder(BorderFactory
378         .createTitledBorder("Chorus"));
379     choruseffect.setOpaque(false);
380     choruseffect.setLayout(new GridBagLayout());
381
382     c.gridx = 0;
383     c.gridy = 0;
384     choruseffect.add(new JLabel("Level"),c);
385     c.gridx = 1;
386     choruseffect.add(createMidiControlSlider(93, 0),c);
387
388     c.gridx = 0;
389     c.gridy = 1;
390     choruseffect.add(new JLabel("Type"),c);
391     c.gridx = 1;
392     choruseffect.add(createChorusTypeCombobox(),c);
393
394     c.gridx = 0;
395     c.gridy = 2;
396     choruseffect.add(new JLabel("Mod_Rate"),c);
397     c.gridx = 1;
398     choruseffect.add(createChorusModRateSlider(),c);
399
400     c.gridx = 0;
401     c.gridy = 3;
402     choruseffect.add(new JLabel("Mod_Depth"),c);
403     c.gridx = 1;
404     choruseffect.add(createChorusModDepthSlider(),c);
405
406     c.gridx = 0;
407     c.gridy = 4;
408     choruseffect.add(new JLabel("Feedback"),c);
409     c.gridx = 1;
410     choruseffect.add(createChorusFeedbackSlider(),c);
411

```

```

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

```
1 import java.awt.BorderLayout;
2 import java.awt.Color;
3 import java.awt.Dimension;
4 import java.awt.FlowLayout;
5 import java.awt.Point;
6 import java.awt.event.ActionEvent;
7 import java.awt.event.ActionListener;
8
9 import javax.sound.midi.Patch;
10 import javax.sound.midi.Receiver;
11 import javax.sound.midi.Soundbank;
12 import javax.sound.midi.Synthesizer;
13 import javax.swing.BoxLayout;
14 import javax.swing.JApplet;
15 import javax.swing.JComboBox;
16 import javax.swing.JLabel;
17 import javax.swing.JPanel;
18 import javax.swing.JScrollPane;
19 import javax.swing.SwingUtilities;
20
21 import com.sun.media.sound.SoftSynthesizer;
22
23 public class SimpleApplet1 extends JApplet {
24
25     private static final long serialVersionUID = 1L;
26
27     private Synthesizer synth = new SoftSynthesizer();
28
29     private Receiver recv;
30
31     JPanel firstpanel;
32     JLabel infolabel;
33     String error_text = "";
34
35     public void destroy() {
36         synth.close();
37     }
38
39     public void init() {
40         Runnable runnable = new Runnable() {
41             public void run() {
42                 try {
43
44                     SwingUtilities.invokeAndWait(new Runnable() {
45                         public void run() {
46                             firstpanel = new JPanel();
47                             firstpanel.setBackground(Color.WHITE);
48                             firstpanel.setLayout(new FlowLayout());
49                             add(firstpanel);
50
51                             infolabel = new JLabel(
52                                 "Loading synthesizer, please wait...");
53
54                             firstpanel.add(infolabel);
55
56                             validate();
57                             invalidate();
58                         }
59                     });
60
```

```

61         synth.getDefaultSoundbank();
62         synth.open();
63         recv = synth.getReceiver();
64
65         SwingUtilities.invokeLater(new Runnable() {
66             public void run() {
67                 createGUI();
68                 validate();
69                 invalidate();
70             }
71         });
72
73     } catch (Exception e) {
74         e.printStackTrace();
75     }
76 }
77 };
78 new Thread(runnable).start();
79 }
80
81 public void createGUI() {
82     remove(firstpanel);
83     JPanel toppanel = new JPanel();
84     toppanel.setBackground(Color.WHITE);
85     toppanel.setLayout(new FlowLayout());
86     add(toppanel);
87
88     JPanel boxpanel = new JPanel();
89     boxpanel.setOpaque(false);
90     boxpanel.setLayout(new BoxLayout(boxpanel, BoxLayout.Y_AXIS));
91     toppanel.add(boxpanel);
92
93     Soundbank sbk = synth.getDefaultSoundbank();
94     String[] instruments = new String[128];
95     for (int i = 0; i < instruments.length; i++) {
96         instruments[i] = i + "_" + sbk.getInstrument(new Patch(0, i)).getName();
97     }
98     final JComboBox instrumentcombobox = new JComboBox(instruments);
99     instrumentcombobox.addActionListener(new ActionListener()
100     {
101         public void actionPerformed(ActionEvent e) {
102             int pgm = instrumentcombobox.getSelectedIndex();
103             synth.getChannels()[0].programChange(pgm);
104         }
105     });
106     JPanel instrumentpanel = new JPanel();
107     instrumentpanel.setLayout(new FlowLayout(FlowLayout.LEFT));
108     instrumentpanel.setOpaque(false);
109     instrumentpanel.add(new JLabel("Instrument:"));
110     instrumentpanel.add(instrumentcombobox);
111     boxpanel.add(instrumentpanel);
112
113     Dimension vdim12 = new Dimension(1000, 50);
114     VirtualKeyboard12 vkeyboard12 = new VirtualKeyboard12();
115     vkeyboard12.setSize(vdim12);
116     vkeyboard12.setPreferredSize(vdim12);
117     vkeyboard12.setMinimumSize(vdim12);
118     vkeyboard12.setMaximumSize(vdim12);
119     vkeyboard12.setChannel(0);
120     vkeyboard12.setReceiver(recv);
121
122     JScrollPane scrollpane12 = new JScrollPane(vkeyboard12);

```

```
123     scrollbar12.setPreferredSize(new Dimension(500, 80));
124     scrollbar12.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

```
1 import java.awt.BorderLayout;
2 import java.awt.Color;
3 import java.awt.Dimension;
4 import java.awt.FlowLayout;
5 import java.awt.Point;
6 import java.awt.event.ActionEvent;
7 import java.awt.event.ActionListener;
8 import java.io.IOException;
9
10 import javax.sound.midi.InvalidMidiDataException;
11 import javax.sound.midi.Patch;
12 import javax.sound.midi.Receiver;
13 import javax.sound.midi.ShortMessage;
14 import javax.sound.midi.Soundbank;
15 import javax.sound.midi.Synthesizer;
16 import javax.sound.midi.SysexMessage;
17 import javax.swing.BorderFactory;
18 import javax.swing.BoxLayout;
19 import javax.swing.JApplet;
20 import javax.swing.JComboBox;
21 import javax.swing.JLabel;
22 import javax.swing.JPanel;
23 import javax.swing.JScrollPane;
24 import javax.swing.SwingUtilities;
25
26 import com.sun.media.sound.SoftSynthesizer;
27
28 public class TuningApplet1 extends JApplet {
29
30     private static final long serialVersionUID = 1L;
31
32     private Synthesizer synth = new SoftSynthesizer();
33
34     private Receiver recv;
35
36     JPanel firstpanel;
37     JLabel infolabel;
38     String error_text = "";
39
40     public void destroy() {
41         synth.close();
42     }
43
44     public void init() {
45         Runnable runnable = new Runnable() {
46             public void run() {
47                 try {
48
49                     SwingUtilities.invokeAndWait(new Runnable() {
50                         public void run() {
51                             firstpanel = new JPanel();
52                             firstpanel.setBackground(Color.WHITE);
53                             firstpanel.setLayout(new FlowLayout());
54                             add(firstpanel);
55
56                             infolabel = new JLabel(
57                                 "Loading synthesizer, please wait...");
58
59                             firstpanel.add(infolabel);
60
```

```

61         validate();
62         invalidate();
63     }
64 });
65
66     synth.getDefaultSoundbank();
67     synth.open();
68     recv = synth.getReceiver();
69
70     SwingUtilities.invokeLater(new Runnable() {
71         public void run() {
72             createGUI();
73             validate();
74             invalidate();
75         }
76     });
77
78     } catch (Exception e) {
79         e.printStackTrace();
80     }
81 }
82 };
83 new Thread(runnable).start();
84 }
85
86 public static void sendTuningChange(Receiver recv, int channel,
87     int tuningpreset) throws InvalidMidiDataException {
88     // Data Entry
89     ShortMessage sm1 = new ShortMessage();
90     sm1.setMessage(ShortMessage.CONTROL_CHANGE, channel, 0x64, 03);
91     ShortMessage sm2 = new ShortMessage();
92     sm2.setMessage(ShortMessage.CONTROL_CHANGE, channel, 0x65, 00);
93     // Tuning program 19
94     ShortMessage sm3 = new ShortMessage();
95     sm3
96         .setMessage(ShortMessage.CONTROL_CHANGE, channel, 0x06,
97             tuningpreset);
98
99     // Data Increment
100    ShortMessage sm4 = new ShortMessage();
101    sm4.setMessage(ShortMessage.CONTROL_CHANGE, channel, 0x60, 0x7F);
102    // Data Decrement
103    ShortMessage sm5 = new ShortMessage();
104    sm5.setMessage(ShortMessage.CONTROL_CHANGE, channel, 0x61, 0x7F);
105
106    recv.send(sm1, -1);
107    recv.send(sm2, -1);
108    recv.send(sm3, -1);
109    recv.send(sm4, -1);
110    recv.send(sm5, -1);
111 }
112
113 public static void sendTunings(Receiver recv, int bank, int preset,
114     String name, double[] tunings) throws IOException,
115     InvalidMidiDataException {
116     int[] itunings = new int[128];
117     for (int i = 0; i < itunings.length; i++) {
118         itunings[i] = (int) (tunings[i] * 16384.0 / 100.0);
119     }
120     SysexMessage msg = UniversalSysExBuilder.MidiTuningStandard
121         .keyBasedTuningDump(UniversalSysExBuilder.ALL_DEVICES, bank,
122             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

```

```

185 Dimension vdim19 = new Dimension(627, 50);
186 VirtualKeyboard19 vkeyboard19 = new VirtualKeyboard19();
187 vkeyboard19.setSize(vdim19);
188 vkeyboard19.setPreferredSize(vdim19);
189 vkeyboard19.setMinimumSize(vdim19);
190 vkeyboard19.setMaximumSize(vdim19);
191 vkeyboard19.setReceiver(recv);
192
193 Dimension vdim12 = new Dimension(1000, 50);
194 VirtualKeyboard12 vkeyboard12 = new VirtualKeyboard12();
195 vkeyboard12.setSize(vdim12);
196 vkeyboard12.setPreferredSize(vdim12);
197 vkeyboard12.setMinimumSize(vdim12);
198 vkeyboard12.setMaximumSize(vdim12);
199 vkeyboard12.setChannel(1);
200 vkeyboard12.setReceiver(recv);
201
202 Dimension vdim7 = new Dimension(1700, 50);
203 VirtualKeyboard7 vkeyboard7 = new VirtualKeyboard7();
204 vkeyboard7.setSize(vdim7);
205 vkeyboard7.setPreferredSize(vdim7);
206 vkeyboard7.setMinimumSize(vdim7);
207 vkeyboard7.setMaximumSize(vdim7);
208 vkeyboard7.setChannel(2);
209 vkeyboard7.setReceiver(recv);
210
211 Dimension vdim5 = new Dimension(1700, 50);
212 VirtualKeyboard5 vkeyboard5 = new VirtualKeyboard5();
213 vkeyboard5.setSize(vdim5);
214 vkeyboard5.setPreferredSize(vdim5);
215 vkeyboard5.setMinimumSize(vdim5);
216 vkeyboard5.setMaximumSize(vdim5);
217 vkeyboard5.setChannel(3);
218 vkeyboard5.setReceiver(recv);
219
220 JScrollPane scrollpane19 = new JScrollPane(vkeyboard19);
221 scrollpane19.setPreferredSize(new Dimension(500, 80));
222 scrollpane19.getViewport().setViewPosition(new Point(107, 0));
223
224 JScrollPane scrollpane12 = new JScrollPane(vkeyboard12);
225 scrollpane12.setPreferredSize(new Dimension(500, 80));
226 scrollpane12.getViewport().setViewPosition(new Point(200, 0));
227
228 JScrollPane scrollpane7 = new JScrollPane(vkeyboard7);
229 scrollpane7.setPreferredSize(new Dimension(500, 80));
230 scrollpane7.getViewport().setViewPosition(new Point(320, 0));
231
232 JScrollPane scrollpane5 = new JScrollPane(vkeyboard5);
233 scrollpane5.setPreferredSize(new Dimension(500, 80));
234 scrollpane5.getViewport().setViewPosition(new Point(320, 0));
235
236 JPanel panel19 = new JPanel(new BorderLayout());
237 panel19.setOpaque(false);
238 panel19.setBorder(BorderFactory
239     .createTitledBorder("19_equal_temperament_(19-TET)"));
240 panel19.add(scrollpane19);
241 boxpanel.add(panel19);
242
243 JPanel panel12 = new JPanel(new BorderLayout());
244 panel12.setOpaque(false);
245 panel12.setBorder(BorderFactory
246     .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

```
1 import java.awt.BorderLayout;
2 import java.awt.Color;
3 import java.awt.Dimension;
4 import java.awt.FlowLayout;
5 import java.awt.Point;
6 import java.awt.event.ActionEvent;
7 import java.awt.event.ActionListener;
8 import java.io.IOException;
9
10 import javax.sound.midi.InvalidMidiDataException;
11 import javax.sound.midi.Patch;
12 import javax.sound.midi.Receiver;
13 import javax.sound.midi.ShortMessage;
14 import javax.sound.midi.Soundbank;
15 import javax.sound.midi.Synthesizer;
16 import javax.sound.midi.SysexMessage;
17 import javax.swing.BorderFactory;
18 import javax.swing.BoxLayout;
19 import javax.swing.JApplet;
20 import javax.swing.JComboBox;
21 import javax.swing.JLabel;
22 import javax.swing.JPanel;
23 import javax.swing.JScrollPane;
24 import javax.swing.SwingUtilities;
25
26 import com.sun.media.sound.SoftSynthesizer;
27
28 public class TuningApplet2 extends JApplet {
29
30     private static final long serialVersionUID = 1L;
31
32     private Synthesizer synth = new SoftSynthesizer();
33
34     private Receiver recv;
35
36     JPanel firstpanel;
37     JLabel infolabel;
38     String error_text = "";
39
40     public void destroy() {
41         synth.close();
42     }
43
44     public void init() {
45         Runnable runnable = new Runnable() {
46             public void run() {
47                 try {
48
49                     SwingUtilities.invokeAndWait(new Runnable() {
50                         public void run() {
51                             firstpanel = new JPanel();
52                             firstpanel.setBackground(Color.WHITE);
53                             firstpanel.setLayout(new FlowLayout());
54                             add(firstpanel);
55
56                             infolabel = new JLabel(
57                                 "Loading synthesizer, please wait...");
58
59                             firstpanel.add(infolabel);
60
```

```

61         validate();
62         invalidate();
63     }
64 });
65
66     synth.getDefaultSoundbank();
67     synth.open();
68     recv = synth.getReceiver();
69
70     SwingUtilities.invokeLater(new Runnable() {
71         public void run() {
72             createGUI();
73             validate();
74             invalidate();
75         }
76     });
77
78     } catch (Exception e) {
79         e.printStackTrace();
80     }
81 }
82 };
83 new Thread(runnable).start();
84 }
85
86 public static void sendTuningChange(Receiver recv, int channel,
87     int tuningpreset) throws InvalidMidiDataException {
88     // Data Entry
89     ShortMessage sm1 = new ShortMessage();
90     sm1.setMessage(ShortMessage.CONTROL_CHANGE, channel, 0x64, 03);
91     ShortMessage sm2 = new ShortMessage();
92     sm2.setMessage(ShortMessage.CONTROL_CHANGE, channel, 0x65, 00);
93     // Tuning program 19
94     ShortMessage sm3 = new ShortMessage();
95     sm3
96         .setMessage(ShortMessage.CONTROL_CHANGE, channel, 0x06,
97             tuningpreset);
98
99     // Data Increment
100    ShortMessage sm4 = new ShortMessage();
101    sm4.setMessage(ShortMessage.CONTROL_CHANGE, channel, 0x60, 0x7F);
102    // Data Decrement
103    ShortMessage sm5 = new ShortMessage();
104    sm5.setMessage(ShortMessage.CONTROL_CHANGE, channel, 0x61, 0x7F);
105
106    recv.send(sm1, -1);
107    recv.send(sm2, -1);
108    recv.send(sm3, -1);
109    recv.send(sm4, -1);
110    recv.send(sm5, -1);
111 }
112
113 public static void sendScaleOctaveTunings(Receiver recv, int bank, int preset,
114     String name, double[] tunings) throws IOException,
115     InvalidMidiDataException {
116     int[] itunings = new int[12];
117     for (int i = 0; i < itunings.length; i++) {
118         itunings[i] = (int) (tunings[i] * 8192.0 / 100.0);
119     }
120     SysexMessage msg = UniversalSysExBuilder.MidiTuningStandard
121         .scaleOctaveTuningDump2ByteForm(UniversalSysExBuilder.ALL_DEVICES, bank, preset,
122             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,

```

```

184         90.22-100.0,
185         203.91-200.0,
186         294.13-300.0,
187         407.82-400.0,
188         498.04-500.0,
189         611.73-600.0,
190         701.96-700.0,
191         792.18-800.0,
192         905.87-900.0,
193         996.09-1000.0,
194         1109.78-1100.0});
195     sendScaleOctaveTunings(recv, 0, 101, "Pyth", tunings);
196     sendTuningChange(recv, 1, 101);
197
198     tunings = new double[] {
199         0.0,
200         76.0-100.0,
201         193.2-200.0,
202         310.3-300.0,
203         386.3-400.0,
204         503.4-500.0,
205         579.5-600.0,
206         696.6-700.0,
207         772.6-800.0,
208         889.7-900.0,
209         1006.8-1000.0,
210         1082.9-1100.0});
211     sendScaleOctaveTunings(recv, 0, 102, "Meantone", tunings);
212     sendTuningChange(recv, 2, 102);
213
214     tunings = new double[] {
215         0-0,
216         90-100,
217         192-200,
218         294-300,
219         390-400,
220         498-500,
221         588-600,
222         696-700,
223         792-800,
224         888-900,
225         996-1000,
226         1092-1100});
227     sendScaleOctaveTunings(recv, 0, 103, "WellTemp", tunings);
228     sendTuningChange(recv, 3, 103);
229
230
231
232 } catch (Exception e1) {
233     e1.printStackTrace();
234     return;
235 }
236
237 Dimension vdim19 = new Dimension(627, 50);
238 VirtualKeyboard19 vkeyboard19 = new VirtualKeyboard19();
239 vkeyboard19.setSize(vdim19);
240 vkeyboard19.setPreferredSize(vdim19);
241 vkeyboard19.setMinimumSize(vdim19);
242 vkeyboard19.setMaximumSize(vdim19);
243 vkeyboard19.setReceiver(recv);
244
245 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);
```

```

61 // Tuning program 19
62 ShortMessage sm3 = new ShortMessage();
63 sm3
64     .setMessage(ShortMessage.CONTROL_CHANGE, channel, 0x06,
65                 tuningpreset);
66
67 // Data Increment
68 ShortMessage sm4 = new ShortMessage();
69 sm4.setMessage(ShortMessage.CONTROL_CHANGE, channel, 0x60, 0x7F);
70 // Data Decrement
71 ShortMessage sm5 = new ShortMessage();
72 sm5.setMessage(ShortMessage.CONTROL_CHANGE, channel, 0x61, 0x7F);
73
74 recv.send(sm1, -1);
75 recv.send(sm2, -1);
76 recv.send(sm3, -1);
77 recv.send(sm4, -1);
78 recv.send(sm5, -1);
79 }
80
81 public void retune(int bank, int preset, int note, double tuning)
82     throws IOException, InvalidMidiDataException {
83     int[] key_numbers = new int[1];
84     int[] key_tunings = new int[1];
85
86     key_numbers[0] = note;
87     key_tunings[0] = (int) (tuning * 16384.0 / 100.0);
88
89     SysexMessage msg = UniversalSysExBuilder.MidiTuningStandard
90         .singleNoteTuningChange(UniversalSysExBuilder.ALL_DEVICES,
91                                true, bank, preset, key_numbers, key_tunings);
92     recv.send(msg, -1);
93 }
94
95 public class SoundPainter extends JComponent {
96
97     private static final long serialVersionUID = 1L;
98
99     boolean was_prev_selected = false;
100
101     int was_prev_selected_x;
102
103     int was_prev_selected_y;
104
105     int activenote = -1;
106
107     MidiChannel channel;
108
109     boolean[] activenotes = new boolean[32];
110
111     int[] activenotes_x = new int[32];
112
113     int[] activenotes_y = new int[32];
114
115     public SoundPainter() {
116         channel = synth.getChannels()[0];
117         addMouseListener(new MouseListener() {
118
119             public void mousePressed(MouseEvent e) {
120
121                 // Check if we clicked on existing note
122                 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     cx += nw;
265 }
266 cx = 0;
267 nw = w / 128f;
268 float black_note_width = nw;
269 for (int i = 0; i < 128; i++) {
270     int b = i % 12;
271     boolean a = (b == 1 || b == 3 | b == 6 | b == 8 | b == 10);
272     if (!a) {
273
274     } else {
275         rect.setRect(cx, th - th * 4.0 / 7.0, black_note_width,
276                     th * 4.0 / 7.0);
277         g2.setColor(Color.BLACK);
278         g2.fill(rect);
279         g2.setColor(Color.BLACK);
280         g2.draw(rect);
281     }
282     cx += nw;
283 }
284
285 // Draw harmonics
286 for (int j = 0; j < 12; j++) {
287
288     int ra = j;
289     int rb = j-1;
290     if(j == 0) { ra = 16; rb = 15; }
291     if(j == 1) { ra = 9; rb = 8; }
292     if(j == 2) { ra = 6; rb = 5; }
293     if(j == 3) { ra = 5; rb = 4; }
294     if(j == 4) { ra = 4; rb = 3; }
295     if(j == 5) { ra = 7; rb = 5; }
296     if(j == 6) { ra = 3; rb = 2; }
297     if(j == 7) { ra = 8; rb = 5; }
298     if(j == 8) { ra = 5; rb = 3; }
299     if(j == 9) { ra = 7; rb = 4; }
300     if(j == 10) { ra = 15; rb = 8; }
301     if(j == 11) { ra = 2; rb = 1; }
302
303     double ratio = ((double)ra) / ((double)rb);
304     double cent = 12.0 * Math.log(ratio) / Math.log(2.0);
305     int jx = (int)(cent * (getWidth() / 128.0));
306     String s_r = ra + "/" + rb;
307
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,
323                        activenotes_y[i]+3);
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
359 public void init() {
360     Runnable runnable = new Runnable() {
361         public void run() {
362             try {
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

```

```

370         infolabel = new JLabel(
371             "Loading synthesizer, please wait...");
372
373         firstpanel.add(infolabel);
374
375         validate();
376         invalidate();
377     }
378 });
379
380 synth.getDefaultSoundbank();
381 synth.open();
382 recv = synth.getReceiver();
383
384 SwingUtilities.invokeLater(new Runnable() {
385     public void run() {
386         createGUI();
387         validate();
388         invalidate();
389     }
390 });
391
392     } catch (Exception e) {
393         e.printStackTrace();
394     }
395 }
396 };
397 new Thread(runnable).start();
398 }
399
400 public void createGUI() {
401     remove(firstpanel);
402     JPanel toppanel = new JPanel();
403     toppanel.setBackground(Color.WHITE);
404     toppanel.setLayout(new FlowLayout());
405     add(toppanel);
406
407     JPanel boxpanel = new JPanel();
408     boxpanel.setOpaque(false);
409     boxpanel.setLayout(new BoxLayout(boxpanel, BoxLayout.Y_AXIS));
410     toppanel.add(boxpanel);
411
412     final SoundPainter soundpainter = new SoundPainter();
413
414     Soundbank sbk = synth.getDefaultSoundbank();
415     String[] instruments = new String[128];
416     for (int i = 0; i < instruments.length; i++) {
417         instruments[i] = i + " "
418             + sbk.getInstrument(new Patch(0, i)).getName();
419     }
420     final JComboBox instrumentcombobox = new JComboBox(instruments);
421     instrumentcombobox.addActionListener(new ActionListener() {
422         public void actionPerformed(ActionEvent e) {
423             int pgm = instrumentcombobox.getSelectedIndex();
424             synth.getChannels()[0].programChange(pgm);
425             soundpainter.allOff();
426         }
427     });
428
429     try {
430         sendTuningChange(recv, 0, 99);
431     } catch (Exception e1) {

```

```

432     }
433     instrumentcombobox.setSelectedIndex(48);
434
435     JPanel instrumentpanel = new JPanel();
436     instrumentpanel.setLayout(new FlowLayout(FlowLayout.LEFT));
437     instrumentpanel.setOpaque(false);
438     instrumentpanel.add(new JLabel("Instrument:"));
439     instrumentpanel.add(instrumentcombobox);
440     boxpanel.add(instrumentpanel);
441
442     Dimension vdim12 = new Dimension(3000, 50);
443     soundpainter.setSize(vdim12);
444     soundpainter.setPreferredSize(vdim12);
445     soundpainter.setMinimumSize(vdim12);
446     soundpainter.setMaximumSize(vdim12);
447     // soundpainter.setChannel(0);
448     // soundpainter.setReceiver(recv);
449
450     JScrollPane scrollpane12 = new JScrollPane(soundpainter);
451     scrollpane12.setPreferredSize(new Dimension(500, 160));
452     scrollpane12.getViewPort().setViewPosition(new Point(1200, 0));
453
454     JPanel panel12 = new JPanel(new BorderLayout());
455     panel12.setOpaque(false);
456     panel12.add(scrollpane12);
457     boxpanel.add(panel12);
458
459 }
460
461 }

```

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,

```

```

123         new byte[] { (byte) value });
124     }
125
126     public static SysexMessage setGlobalParameter(int targetDevice,
127         short[] slotpath, byte[] parameter, byte[] value)
128         throws IOException, InvalidMidiDataException {
129         ByteArrayOutputStream baos = new ByteArrayOutputStream();
130         baos.write(UNIVERSAL_REALTIME_SYSEX_HEADER);
131         baos.write((byte) targetDevice);
132         baos.write(DEVICE_CONTROL);
133         baos.write(GLOBAL_PARAMETER_CONTROL);
134         baos.write((byte) slotpath.length);
135         baos.write((byte) parameter.length);
136         baos.write((byte) value.length);
137         for (int i = 0; i < slotpath.length; i++) {
138             short x = slotpath[i];
139             baos.write((byte) (x >>> 8));
140             baos.write((byte) x);
141         }
142         baos.write(parameter);
143         baos.write(value);
144         baos.write(EOX);
145         SysexMessage sysex = new SysexMessage();
146         byte[] data = baos.toByteArray();
147         sysex.setMessage(data, data.length);
148         return sysex;
149     }
150
151     public static class ReverbEffect {
152
153         public static final int REVERB_TYPE_SMALL_ROOM = 0;
154
155         public static final int REVERB_TYPE_MEDIUM_ROOM = 1;
156
157         public static final int REVERB_TYPE_LARGE_ROOM = 2;
158
159         public static final int REVERB_TYPE_MEDIUM_HALL = 3;
160
161         public static final int REVERB_TYPE_LARGE_HALL = 4;
162
163         public static final int REVERB_TYPE_PLATE = 8;
164
165         private static final short[] SLOTPATH_EFFECT_REVERB = new short[] { (short) 0x0101 };
166
167         private static final byte[] REVERB_TYPE = new byte[] { (byte) 0x00 };
168
169         private static final byte[] REVERB_TIME = new byte[] { (byte) 0x01 };
170
171         public static SysexMessage setReverbType(int targetDevice,
172             int reverbType) throws IOException,
173             InvalidMidiDataException {
174             return setGlobalParameter(targetDevice, SLOTPATH_EFFECT_REVERB,
175                 REVERB_TYPE, reverbType);
176         }
177
178         public static SysexMessage setReverbTime(int targetDevice,
179             int reverbTime) throws IOException,
180             InvalidMidiDataException {
181             return setGlobalParameter(targetDevice, SLOTPATH_EFFECT_REVERB,
182                 REVERB_TIME, reverbTime);
183         }
184     }

```



```

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

```

```

431     baos.write((byte) targetDevice);
432     baos.write(MIDI_TUNING_STANDARD);
433     baos.write(SCALE_OCTAVE_TUNING_2BYTE_FORM);
434     int channelmask = 0;
435     for (int i = 0; i < 2; i++) {
436         if (channels[i + 14])
437             channelmask += 1 << i;
438     }
439     baos.write((byte) channelmask);
440     channelmask = 0;
441     for (int i = 0; i < 7; i++) {
442         if (channels[i + 7])
443             channelmask += 1 << i;
444     }
445     baos.write((byte) channelmask);
446     channelmask = 0;
447     for (int i = 0; i < 7; i++) {
448         if (channels[i])
449             channelmask += 1 << i;
450     }
451     baos.write((byte) channelmask);
452     for (int i = 0; i < 12; i++) {
453         int t = tuning[i] + 8192;
454         baos.write((byte) (t / 128));
455         baos.write((byte) (t % 128));
456     }
457     baos.write(EOX);
458     SysexMessage sysex = new SysexMessage();
459     byte[] data = baos.toByteArray();
460     sysex.setMessage(data, data.length);
461     return sysex;
462 }
463
464 private static void setTuningChecksum(byte[] data) {
465     int x = data[1] & 0xFF;
466     for (int i = 2; i < data.length - 2; i++)
467         x = x ^ (data[i] & 0xFF);
468     data[data.length - 2] = (byte) (x & 127);
469 }
470
471 public static SysexMessage scaleOctaveTuningDump1ByteForm(
472     int targetDevice, int bank, int preset, String name,
473     int[] tuning) throws IOException, InvalidMidiDataException {
474     ByteArrayOutputStream baos = new ByteArrayOutputStream();
475     baos.write(UNIVERSAL_NON_REALTIME_SYSEX_HEADER);
476     baos.write((byte) targetDevice);
477     baos.write(MIDI_TUNING_STANDARD);
478     baos.write(SCALE_OCTAVE_TUNING_DUMP_1BYTE_FORM);
479     baos.write((byte) bank);
480     baos.write((byte) preset);
481     if (name.length() > 16)
482         name = name.substring(0, 16);
483     byte[] namebytes = name.getBytes("ASCII");
484     baos.write(namebytes);
485     byte space_char = " ".getBytes()[0];
486     for (int i = namebytes.length; i < 16; i++)
487         baos.write(space_char);
488     for (int i = 0; i < 12; i++) {
489         baos.write((byte) (tuning[i] + 64));
490     }
491     baos.write(0);
492     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 = "zsxdcvgbhnmjq2w3er5t6y7ui9o0p".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         {
```

```

61     int xb = x - (int)(oct * 7 * nw);
62     float cx = 0;
63     float black_note_width = nw * 0.7f;
64     for (int b = 0; b < 12; b++) {
65         boolean a = (b==1 || b==3 || b==6 || b==8 || b==10);
66         if(!a)
67         {
68             cx += nw;
69         }
70         else
71         {
72             float cstart = cx - (black_note_width/2);
73             float cend = cstart + black_note_width;
74             if(xb > cstart && xb < cend)
75             {
76                 return oct*12 + b;
77             }
78         }
79     }
80
81 }
82
83 if(n < 0) n = 0;
84 if(n > 127) n = 127;
85 return n;
86 }
87
88 private void allKeyboardKeyOff()
89 {
90     for (int i = 0; i < keyDown.length; i++) {
91         if(keyDown[i])
92             if((i + lowestKey) < 128)
93             {
94                 ShortMessage sm = new ShortMessage();
95                 try {
96                     sm.setMessage(ShortMessage.NOTE_OFF, channel, (i + lowestKey), 0);
97                     if(recv != null)
98                         recv.send(sm, -1);
99                     send(sm, -1);
100                 } catch (InvalidMidiDataException e1) {
101                     e1.printStackTrace();
102                 }
103                 keyDown[i] = false;
104             }
105     }
106 }
107
108 public void setChannel(int c) {
109     channel = c;
110 }
111
112 public void setVelocity(int v) {
113     velocity = v;
114 }
115
116 public VirtualKeyboard12() {
117     super();
118     setFocusable(true);
119
120     addMouseListener(new MouseAdapter()
121     {
122         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             ;
131             if(recv != null)
132                 recv.send(sm, -1);
133             send(sm, -1);
134         } catch (InvalidMidiDataException e1) {
135             e1.printStackTrace();
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),
170                                 velocity);
171                             if(recv != null)
172                                 recv.send(sm, -1);
173                             send(sm, -1);
174                         } catch (InvalidMidiDataException e1) {
175                             e1.printStackTrace();
176                         }
177                         keyDown[i] = true;
178                     }
179                 return;
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                         ;
195                         if(recv != null)
196                             recv.send(sm, -1);
197                         send(sm, -1);
198                     } catch (InvalidMidiDataException e1) {
199                         e1.printStackTrace();
200                     }
201                     keyDown[i] = false;
202                 }
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);

```

```

244 Graphics2D g2 = (Graphics2D)g;
245 g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
246     RenderingHints.VALUE_ANTIALIAS_ON);
247 g2.setRenderingHint(RenderingHints.KEY_FRACTIONALMETRICS,
248     RenderingHints.VALUE_FRACTIONALMETRICS_ON);
249
250 int w = getWidth();
251 int h = getHeight();
252
253 float nw = w / 75f;
254 float cx = 0;
255 Rectangle2D rect = new Rectangle2D.Double();
256 for (int i = 0; i < 128; i++) {
257     int b = i % 12;
258     boolean a = (b==1 || b==3 || b==6 || b==8 || b==10);
259     if(!a)
260     {
261         rect.setRect(cx, 0, nw, h);
262         if(noteDown[i])
263             g2.setColor(new Color(0.8f, 0.8f, 0.95f));
264         else
265             g2.setColor(Color.WHITE);
266         g2.fill(rect);
267         g2.setColor(Color.BLACK);
268         g2.draw(rect);
269
270         if(hasFocus() && (i >= lowestKey))
271             if(i >= lowestKey)
272             {
273                 if(i - lowestKey < virtualKeys.length)
274                 {
275                     g2.setColor(Color.GRAY);
276                     char k = virtualKeys[i - lowestKey];
277                     g2.drawString("" + k, cx + 2, h - 4);
278                 }
279             }
280
281         cx += nw;
282     }
283 }
284 cx = 0;
285 float black_note_width = nw * 0.7f;
286 for (int i = 0; i < 128; i++) {
287     int b = i % 12;
288     boolean a = (b==1 || b==3 || b==6 || b==8 || b==10);
289     if(!a)
290     {
291         cx += nw;
292     }
293     else
294     {
295         rect.setRect(cx - (black_note_width/2), 0, black_note_width, h*4.0/7.0);
296         if(noteDown[i])
297             g2.setColor(new Color(0.8f, 0.8f, 0.95f));
298         else
299             g2.setColor(Color.BLACK);
300         g2.fill(rect);
301         g2.setColor(Color.BLACK);
302         g2.draw(rect);
303
304         if(hasFocus() && (i >= lowestKey))
305             if(i >= lowestKey)

```

```

306         {
307             if(i - lowestKey < virtualKeys.length)
308             {
309                 g2.setColor(Color.LIGHT_GRAY);
310                 char k = virtualKeys[i - lowestKey];
311                 g2.drawString(" " + k, cx - (black_note_width/2) + 1, (h*4.0f/7.0f) -
312                             3);
313             }
314         }
315     }
316 }
317
318
319 public void close() {
320
321
322 }
323
324 public void send(MidiMessage message, long timeStamp) {
325     if(message instanceof ShortMessage)
326     {
327         ShortMessage sm = (ShortMessage)message;
328         if(sm.getChannel() == channel)
329         if(sm.getCommand() == ShortMessage.NOTE_ON
330            || sm.getCommand() == ShortMessage.NOTE_OFF)
331         {
332             noteDown[sm.getData1()] =
333                 (sm.getCommand() == ShortMessage.NOTE_ON) && (sm.getData2() != 0);
334             repaint();
335         }
336     }
337 }
338
339 public Receiver getReceiver() {
340     return recv;
341 }
342
343 public void setReceiver(Receiver receiver) {
344     recv = receiver;
345 }
346
347 }

```

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                 ;
149                 if(recv != null)
150                     recv.send(sm, -1);
151                 send(sm, -1);
152             } catch (InvalidMidiDataException e1) {
153                 e1.printStackTrace();
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             public void keyPressed(KeyEvent e) {
177                 char lc = Character.toLowerCase(e.getKeyChar());
178                 for (int i = 0; i < virtualKeys.length; i++) {
179                     if(virtualKeys[i] == lc)
180                     {
181                         if(!keyDown[i])
182                             if((i + lowestKey) < 128)

```

```

184         {
185             ShortMessage sm = new ShortMessage();
186             try {
187                 sm.setMessage(ShortMessage.NOTE_ON, channel, (i + lowestKey),
188                     velocity);
189                 if(recv != null)
190                     recv.send(sm, -1);
191                 send(sm, -1);
192             } catch (InvalidMidiDataException e1) {
193                 e1.printStackTrace();
194             }
195             keyDown[i] = true;
196         }
197     }
198 }
199
200
201 public void keyReleased(KeyEvent e) {
202     char lc = Character.toLowerCase(e.getKeyChar());
203     for (int i = 0; i < virtualKeys.length; i++) {
204         if(virtualKeys[i] == lc)
205         {
206             if(keyDown[i])
207                 if((i + lowestKey) < 128)
208                 {
209                     ShortMessage sm = new ShortMessage();
210                     try {
211                         sm.setMessage(ShortMessage.NOTE_OFF, channel, (i + lowestKey), 0)
212                             ;
213                         if(recv != null)
214                             recv.send(sm, -1);
215                         send(sm, -1);
216                     } catch (InvalidMidiDataException e1) {
217                         e1.printStackTrace();
218                     }
219                     keyDown[i] = false;
220                 }
221             return;
222         }
223     }
224 }
225
226 public void keyTyped(KeyEvent e) {
227     if(e.getKeyChar() == '-')
228     {
229         allKeyboardKeyOff();
230         lowestKey -= 19;
231         if(lowestKey < 0) lowestKey += 19;
232         repaint();
233     }
234     if(e.getKeyChar() == '+')
235     {
236         allKeyboardKeyOff();
237         lowestKey += 19;
238         if(lowestKey > 128) lowestKey -= 19;
239         repaint();
240     }
241 }
242
243 });

```

```

244     addFocusListener(new FocusListener()
245     {
246         public void focusGained(FocusEvent e) {
247             repaint();
248         }
249
250         public void focusLost(FocusEvent e) {
251             repaint();
252             allKeyboardKeyOff();
253         }
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
336                             ) - 3);
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
360                                     ) - 5);
361                             }
362                         }
363                     else
364                     {
365

```

```

366         rect.setRect(cx - (black_note_width/2), h*4.0/7.0/2+1, black_note_width, h
367             *4.0/7.0/2-1);
368         if(noteDown[i])
369             g2.setColor(new Color(0.8f, 0.8f, 0.95f));
370         else
371             g2.setColor(Color.BLACK);
372         g2.fill(rect);
373         g2.setColor(Color.BLACK);
374         g2.draw(rect);
375
376         if(hasFocus() && (i >= lowestKey))
377             if(i >= lowestKey)
378             {
379                 if(i - lowestKey < virtualKeys.length)
380                 {
381                     g2.setColor(Color.LIGHT_GRAY);
382                     char k = virtualKeys[i - lowestKey];
383                     g2.drawString(" " + k, cx - (black_note_width/2) + 1, (h*4.0f/7.0f
384                         ) - 3);
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                 ;
99                 if(recv != null)
100                     recv.send(sm, -1);
101                 send(sm, -1);
102             } catch (InvalidMidiDataException e1) {
103                 e1.printStackTrace();
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),
138                                 velocity);
139                             if(recv != null)
140                                 recv.send(sm, -1);
141                             send(sm, -1);
142                         } catch (InvalidMidiDataException e1) {
143                             e1.printStackTrace();
144                         }
145                         keyDown[i] = true;
146                     }
147                 return;
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                             ;
163                             if(recv != null)
164                                 recv.send(sm, -1);
165                             send(sm, -1);
166                         } catch (InvalidMidiDataException e1) {
167                             e1.printStackTrace();
168                         }
169                         keyDown[i] = false;
170                     }
171                 return;
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;

```



```

182         repaint();
183     }
184     if(e.getKeyChar() == '+')
185     {
186         allKeyboardKeyOff();
187         lowestKey += 7;
188         if(lowestKey > 127) lowestKey -= 7;
189         repaint();
190     }
191 }
192
193 });
194
195 addFocusListener(new FocusListener()
196 {
197     public void focusGained(FocusEvent e) {
198         repaint();
199     }
200
201     public void focusLost(FocusEvent e) {
202         repaint();
203         allKeyboardKeyOff();
204     }
205
206 });
207 }
208
209
210 public void paint(Graphics g) {
211     super.paint(g);
212     Graphics2D g2 = (Graphics2D)g;
213     g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
214         RenderingHints.VALUE_ANTIALIAS_ON);
215     g2.setRenderingHint(RenderingHints.KEY_FRACTIONALMETRICS,
216         RenderingHints.VALUE_FRACTIONALMETRICS_ON);
217
218     int w = getWidth();
219     int h = getHeight();
220
221     float nw = w / 128f;
222     float cx = 0;
223     Rectangle2D rect = new Rectangle2D.Double();
224     for (int i = 0; i < 128; i++) {
225
226         rect.setRect(cx, 0, nw, h);
227         if(noteDown[i])
228             g2.setColor(new Color(0.8f, 0.8f, 0.95f));
229         else
230             g2.setColor(Color.WHITE);
231         g2.fill(rect);
232         g2.setColor(Color.BLACK);
233         g2.draw(rect);
234
235         if(i % 5 == 0)
236             g2.drawString("C", cx + 2, 12);
237
238         if(hasFocus() && (i >= lowestKey))
239             if(i >= lowestKey)
240             {
241                 if(i - lowestKey < virtualKeys.length)
242                 {
243                     g2.setColor(Color.GRAY);

```

```

244         char k = virtualKeys[i - lowestKey];
245         g2.drawString(" " + k, cx + 2, h - 4);
246     }
247 }
248
249     cx += nw;
250 }
251 }
252
253 public void close() {
254
255 }
256
257 public void send(MidiMessage message, long timeStamp) {
258     if(message instanceof ShortMessage)
259     {
260         ShortMessage sm = (ShortMessage)message;
261         if(sm.getChannel() == channel)
262         if(sm.getCommand() == ShortMessage.NOTE_ON
263             || sm.getCommand() == ShortMessage.NOTE_OFF)
264         {
265             noteDown[sm.getData1()] =
266                 (sm.getCommand() == ShortMessage.NOTE_ON) && (sm.getData2() != 0);
267             repaint();
268         }
269     }
270 }
271 }
272
273 public Receiver getReceiver() {
274     return recv;
275 }
276
277 public void setReceiver(Receiver receiver) {
278     recv = receiver;
279 }
280
281 }

```

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 = "zxcvbnmasdfghjkwertyu1234567".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                 ;
99                 if(recv != null)
100                     recv.send(sm, -1);
101                 send(sm, -1);
102             } catch (InvalidMidiDataException e1) {
103                 e1.printStackTrace();
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),
138                                 velocity);
139                             if(recv != null)
140                                 recv.send(sm, -1);
141                             send(sm, -1);
142                         } catch (InvalidMidiDataException e1) {
143                             e1.printStackTrace();
144                         }
145                         keyDown[i] = true;
146                     }
147                 return;
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                             ;
163                             if(recv != null)
164                                 recv.send(sm, -1);
165                             send(sm, -1);
166                         } catch (InvalidMidiDataException e1) {
167                             e1.printStackTrace();
168                         }
169                         keyDown[i] = false;
170                     }
171                 return;
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;

```

```

182         repaint();
183     }
184     if(e.getKeyChar() == '+')
185     {
186         allKeyboardKeyOff();
187         lowestKey += 7;
188         if(lowestKey > 127) lowestKey -= 7;
189         repaint();
190     }
191 }
192
193 });
194
195 addFocusListener(new FocusListener()
196 {
197     public void focusGained(FocusEvent e) {
198         repaint();
199     }
200
201     public void focusLost(FocusEvent e) {
202         repaint();
203         allKeyboardKeyOff();
204     }
205 });
206
207 }
208
209
210 public void paint(Graphics g) {
211     super.paint(g);
212     Graphics2D g2 = (Graphics2D)g;
213     g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
214         RenderingHints.VALUE_ANTIALIAS_ON);
215     g2.setRenderingHint(RenderingHints.KEY_FRACTIONALMETRICS,
216         RenderingHints.VALUE_FRACTIONALMETRICS_ON);
217
218     int w = getWidth();
219     int h = getHeight();
220
221     float nw = w / 128f;
222     float cx = 0;
223     Rectangle2D rect = new Rectangle2D.Double();
224     for (int i = 0; i < 128; i++) {
225
226         rect.setRect(cx, 0, nw, h);
227         if(noteDown[i])
228             g2.setColor(new Color(0.8f, 0.8f, 0.95f));
229         else
230             g2.setColor(Color.WHITE);
231         g2.fill(rect);
232         g2.setColor(Color.BLACK);
233         g2.draw(rect);
234
235         if(i % 7 == 0)
236             g2.drawString("C", cx + 2, 12);
237
238         if(hasFocus() && (i >= lowestKey))
239             if(i >= lowestKey)
240             {
241                 if(i - lowestKey < virtualKeys.length)
242                 {
243                     g2.setColor(Color.GRAY);

```

```

244         char k = virtualKeys[i - lowestKey];
245         g2.drawString(" " + k, cx + 2, h - 4);
246     }
247 }
248
249     cx += nw;
250 }
251 }
252
253 public void close() {
254
255
256 }
257
258 public void send(MidiMessage message, long timeStamp) {
259     if(message instanceof ShortMessage)
260     {
261         ShortMessage sm = (ShortMessage)message;
262         if(sm.getChannel() == channel)
263         if(sm.getCommand() == ShortMessage.NOTE_ON
264             || sm.getCommand() == ShortMessage.NOTE_OFF)
265         {
266             noteDown[sm.getData1()] =
267                 (sm.getCommand() == ShortMessage.NOTE_ON) && (sm.getData2() != 0);
268             repaint();
269         }
270     }
271 }
272
273 public Receiver getReceiver() {
274     return rcv;
275 }
276
277 public void setReceiver(Receiver receiver) {
278     rcv = receiver;
279 }
280
281 }

```

16 com/sun/media/sound/AudioFileSoundbankReader.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.io.ByteArrayOutputStream;
28 import java.io.DataInputStream;
29 import java.io.File;
30 import java.io.IOException;
31 import java.io.InputStream;
32 import java.net.URL;
33
34 import javax.sound.midi.InvalidMidiDataException;
35 import javax.sound.midi.Soundbank;
36 import javax.sound.midi.spi.SoundbankReader;
37 import javax.sound.sampled.AudioInputStream;
38 import javax.sound.sampled.AudioSystem;
39 import javax.sound.sampled.UnsupportedAudioFileException;
40
41 /**
42 * Soundbank reader that uses audio files as soundbanks.
43 *
44 * @author Karl Helgason
45 */
46 public class AudioFileSoundbankReader extends SoundbankReader {
47
48     public Soundbank getSoundbank(URL url)
49         throws InvalidMidiDataException, IOException {
50         try {
51             AudioInputStream ais = AudioSystem.getAudioInputStream(url);
52             Soundbank sbk = getSoundbank(ais);
53             ais.close();
54             return sbk;
55         } catch (UnsupportedAudioFileException e) {
56             return null;
57         } catch (IOException e) {
58             return null;
59         }
60     }
```



```

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

```
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.nio.ByteBuffer;
28 import java.nio.ByteOrder;
29 import java.nio.DoubleBuffer;
30 import java.nio.FloatBuffer;
31
32 import javax.sound.sampled.AudioFormat;
33 import javax.sound.sampled.AudioFormat.Encoding;
34
35 /**
36 * This class is used to convert between 8,16,24,32,32+ bit signed/unsigned
37 * big/little endian fixed/floating point byte buffers and float buffers.
38 *
39 * @author Karl Helgason
40 */
41 public abstract class AudioFloatConverter {
42
43     public static final Encoding PCM_FLOAT = new Encoding("PCM_FLOAT");
44
45     /*****
46     *
47     * LSB Filter, used filter least significant byte in samples arrays.
48     *
49     * Is used filter out data in lsb byte when SampleSizeInBits is not
50     * dividable by 8.
51     *
52     *****/
53
54     private static class AudioFloatLSBFilter extends AudioFloatConverter {
55
56         private AudioFloatConverter converter;
57
58         final private int offset;
59
60         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

```

```

123 /*****
124  *
125  * 64 bit float, little/big-endian
126  *
127  *****/
128
129 // PCM 64 bit float, little-endian
130 private static class AudioFloatConversion64L extends AudioFloatConverter {
131     ByteBuffer bytebuffer = null;
132
133     DoubleBuffer floatbuffer = null;
134
135     double[] double_buff = null;
136
137     public float[] toFloatArray(byte[] in_buff, int in_offset,
138         float[] out_buff, int out_offset, int out_len) {
139         int in_len = out_len * 8;
140         if (bytebuffer == null || bytebuffer.capacity() < in_len) {
141             bytebuffer = ByteBuffer.allocate(in_len).order(
142                 ByteOrder.LITTLE_ENDIAN);
143             floatbuffer = bytebuffer.asDoubleBuffer();
144         }
145         bytebuffer.position(0);
146         floatbuffer.position(0);
147         bytebuffer.put(in_buff, in_offset, in_len);
148         if (double_buff == null
149             || double_buff.length < out_len + out_offset)
150             double_buff = new double[out_len + out_offset];
151         floatbuffer.get(double_buff, out_offset, out_len);
152         int out_offset_end = out_offset + out_len;
153         for (int i = out_offset; i < out_offset_end; i++) {
154             out_buff[i] = (float) double_buff[i];
155         }
156         return out_buff;
157     }
158
159     public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
160         byte[] out_buff, int out_offset) {
161         int out_len = in_len * 8;
162         if (bytebuffer == null || bytebuffer.capacity() < out_len) {
163             bytebuffer = ByteBuffer.allocate(out_len).order(
164                 ByteOrder.LITTLE_ENDIAN);
165             floatbuffer = bytebuffer.asDoubleBuffer();
166         }
167         floatbuffer.position(0);
168         bytebuffer.position(0);
169         if (double_buff == null || double_buff.length < in_offset + in_len)
170             double_buff = new double[in_offset + in_len];
171         int in_offset_end = in_offset + in_len;
172         for (int i = in_offset; i < in_offset_end; i++) {
173             double_buff[i] = in_buff[i];
174         }
175         floatbuffer.put(double_buff, in_offset, in_len);
176         bytebuffer.get(out_buff, out_offset, out_len);
177         return out_buff;
178     }
179 }
180
181 // PCM 64 bit float, big-endian
182 private static class AudioFloatConversion64B extends AudioFloatConverter {
183     ByteBuffer bytebuffer = null;
184

```

```

185 DoubleBuffer floatbuffer = null;
186
187 double[] double_buff = null;
188
189 public float[] toFloatArray(byte[] in_buff, int in_offset,
190     float[] out_buff, int out_offset, int out_len) {
191     int in_len = out_len * 8;
192     if (bytebuffer == null || bytebuffer.capacity() < in_len) {
193         bytebuffer = ByteBuffer.allocate(in_len).order(
194             ByteOrder.BIG_ENDIAN);
195         floatbuffer = bytebuffer.asDoubleBuffer();
196     }
197     bytebuffer.position(0);
198     floatbuffer.position(0);
199     bytebuffer.put(in_buff, in_offset, in_len);
200     if (double_buff == null
201         || double_buff.length < out_len + out_offset)
202         double_buff = new double[out_len + out_offset];
203     floatbuffer.get(double_buff, out_offset, out_len);
204     int out_offset_end = out_offset + out_len;
205     for (int i = out_offset; i < out_offset_end; i++) {
206         out_buff[i] = (float) double_buff[i];
207     }
208     return out_buff;
209 }
210
211 public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
212     byte[] out_buff, int out_offset) {
213     int out_len = in_len * 8;
214     if (bytebuffer == null || bytebuffer.capacity() < out_len) {
215         bytebuffer = ByteBuffer.allocate(out_len).order(
216             ByteOrder.BIG_ENDIAN);
217         floatbuffer = bytebuffer.asDoubleBuffer();
218     }
219     floatbuffer.position(0);
220     bytebuffer.position(0);
221     if (double_buff == null || double_buff.length < in_offset + in_len)
222         double_buff = new double[in_offset + in_len];
223     int in_offset_end = in_offset + in_len;
224     for (int i = in_offset; i < in_offset_end; i++) {
225         double_buff[i] = in_buff[i];
226     }
227     floatbuffer.put(double_buff, in_offset, in_len);
228     bytebuffer.get(out_buff, out_offset, out_len);
229     return out_buff;
230 }
231 }
232
233 /*****
234  *
235  * 32 bit float, little/big-endian
236  *
237  *****/
238
239 // PCM 32 bit float, little-endian
240 private static class AudioFloatConversion32L extends AudioFloatConverter {
241     ByteBuffer bytebuffer = null;
242
243     FloatBuffer floatbuffer = null;
244
245     public float[] toFloatArray(byte[] in_buff, int in_offset,
246         float[] out_buff, int out_offset, int out_len) {

```

```

247     int in_len = out_len * 4;
248     if (bytebuffer == null || bytebuffer.capacity() < in_len) {
249         bytebuffer = ByteBuffer.allocate(in_len).order(
250             ByteOrder.LITTLE_ENDIAN);
251         floatbuffer = bytebuffer.asFloatBuffer();
252     }
253     bytebuffer.position(0);
254     floatbuffer.position(0);
255     bytebuffer.put(in_buff, in_offset, in_len);
256     floatbuffer.get(out_buff, out_offset, out_len);
257     return out_buff;
258 }
259
260 public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
261     byte[] out_buff, int out_offset) {
262     int out_len = in_len * 4;
263     if (bytebuffer == null || bytebuffer.capacity() < out_len) {
264         bytebuffer = ByteBuffer.allocate(out_len).order(
265             ByteOrder.LITTLE_ENDIAN);
266         floatbuffer = bytebuffer.asFloatBuffer();
267     }
268     floatbuffer.position(0);
269     bytebuffer.position(0);
270     floatbuffer.put(in_buff, in_offset, in_len);
271     bytebuffer.get(out_buff, out_offset, out_len);
272     return out_buff;
273 }
274 }
275
276 // PCM 32 bit float, big-endian
277 private static class AudioFloatConversion32B extends AudioFloatConverter {
278     ByteBuffer bytebuffer = null;
279
280     FloatBuffer floatbuffer = null;
281
282     public float[] toFloatArray(byte[] in_buff, int in_offset,
283         float[] out_buff, int out_offset, int out_len) {
284         int in_len = out_len * 4;
285         if (bytebuffer == null || bytebuffer.capacity() < in_len) {
286             bytebuffer = ByteBuffer.allocate(in_len).order(
287                 ByteOrder.BIG_ENDIAN);
288             floatbuffer = bytebuffer.asFloatBuffer();
289         }
290         bytebuffer.position(0);
291         floatbuffer.position(0);
292         bytebuffer.put(in_buff, in_offset, in_len);
293         floatbuffer.get(out_buff, out_offset, out_len);
294         return out_buff;
295     }
296
297     public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
298         byte[] out_buff, int out_offset) {
299         int out_len = in_len * 4;
300         if (bytebuffer == null || bytebuffer.capacity() < out_len) {
301             bytebuffer = ByteBuffer.allocate(out_len).order(
302                 ByteOrder.BIG_ENDIAN);
303             floatbuffer = bytebuffer.asFloatBuffer();
304         }
305         floatbuffer.position(0);
306         bytebuffer.position(0);
307         floatbuffer.put(in_buff, in_offset, in_len);
308         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,

```



```

371         float[] out_buff, int out_offset, int out_len) {
372     int ix = in_offset;
373     int len = out_offset + out_len;
374     for (int ox = out_offset; ox < len; ox++) {
375         out_buff[ox] = ((short) ((in_buff[ix++] & 0xFF) |
376             (in_buff[ix++] << 8))) * (1.0f / 32767.0f);
377     }
378
379     return out_buff;
380 }
381
382 public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
383     byte[] out_buff, int out_offset) {
384     int ox = out_offset;
385     int len = in_offset + in_len;
386     for (int ix = in_offset; ix < len; ix++) {
387         int x = (int) (in_buff[ix] * 32767.0);
388         out_buff[ox++] = (byte) x;
389         out_buff[ox++] = (byte) (x >>> 8);
390     }
391     return out_buff;
392 }
393 }
394
395 // PCM 16 bit, signed, big-endian
396 private static class AudioFloatConversion16SB extends AudioFloatConverter {
397     public float[] toFloatArray(byte[] in_buff, int in_offset,
398         float[] out_buff, int out_offset, int out_len) {
399         int ix = in_offset;
400         int ox = out_offset;
401         for (int i = 0; i < out_len; i++) {
402             out_buff[ox++] = ((short) ((in_buff[ix++] << 8) |
403                 (in_buff[ix++] & 0xFF))) * (1.0f / 32767.0f);
404         }
405         return out_buff;
406     }
407
408     public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
409         byte[] out_buff, int out_offset) {
410         int ix = in_offset;
411         int ox = out_offset;
412         for (int i = 0; i < in_len; i++) {
413             int x = (int) (in_buff[ix++] * 32767.0);
414             out_buff[ox++] = (byte) (x >>> 8);
415             out_buff[ox++] = (byte) x;
416         }
417         return out_buff;
418     }
419 }
420
421 // PCM 16 bit, unsigned, little-endian
422 private static class AudioFloatConversion16UL extends AudioFloatConverter {
423     public float[] toFloatArray(byte[] in_buff, int in_offset,
424         float[] out_buff, int out_offset, int out_len) {
425         int ix = in_offset;
426         int ox = out_offset;
427         for (int i = 0; i < out_len; i++) {
428             int x = (in_buff[ix++] & 0xFF) | ((in_buff[ix++] & 0xFF) << 8);
429             out_buff[ox++] = (x - 32767) * (1.0f / 32767.0f);
430         }
431         return out_buff;
432     }

```

```

433     public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
434                             byte[] out_buff, int out_offset) {
435         int ix = in_offset;
436         int ox = out_offset;
437         for (int i = 0; i < in_len; i++) {
438             int x = 32767 + (int) (in_buff[ix++] * 32767.0);
439             out_buff[ox++] = (byte) x;
440             out_buff[ox++] = (byte) (x >>> 8);
441         }
442         return out_buff;
443     }
444 }
445
446 // PCM 16 bit, unsigned, big-endian
447 private static class AudioFloatConversion16UB extends AudioFloatConverter {
448     public float[] toFloatArray(byte[] in_buff, int in_offset,
449                                float[] out_buff, int out_offset, int out_len) {
450         int ix = in_offset;
451         int ox = out_offset;
452         for (int i = 0; i < out_len; i++) {
453             int x = ((in_buff[ix++] & 0xFF) << 8) | (in_buff[ix++] & 0xFF);
454             out_buff[ox++] = (x - 32767) * (1.0f / 32767.0f);
455         }
456         return out_buff;
457     }
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 > 0x7FFFFFFF)
489                 x -= 0x1000000;
490             out_buff[ox++] = x * (1.0f / (float)0x7FFFFFFF);
491         }
492         return out_buff;
493     }
494 }

```

```

495     public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
496         byte[] out_buff, int out_offset) {
497         int ix = in_offset;
498         int ox = out_offset;
499         for (int i = 0; i < in_len; i++) {
500             int x = (int) (in_buff[ix++] * (float)0x7FFFFFFF);
501             if (x < 0)
502                 x += 0x1000000;
503             out_buff[ox++] = (byte) x;
504             out_buff[ox++] = (byte) (x >>> 8);
505             out_buff[ox++] = (byte) (x >>> 16);
506         }
507         return out_buff;
508     }
509 }
510
511 // PCM 24 bit, signed, big-endian
512 private static class AudioFloatConversion24SB extends AudioFloatConverter {
513     public float[] toFloatArray(byte[] in_buff, int in_offset,
514         float[] out_buff, int out_offset, int out_len) {
515         int ix = in_offset;
516         int ox = out_offset;
517         for (int i = 0; i < out_len; i++) {
518             int x = ((in_buff[ix++] & 0xFF) << 16)
519                 | ((in_buff[ix++] & 0xFF) << 8) | (in_buff[ix++] & 0xFF);
520             if (x > 0x7FFFFFFF)
521                 x -= 0x1000000;
522             out_buff[ox++] = x * (1.0f / (float)0x7FFFFFFF);
523         }
524         return out_buff;
525     }
526
527     public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
528         byte[] out_buff, int out_offset) {
529         int ix = in_offset;
530         int ox = out_offset;
531         for (int i = 0; i < in_len; i++) {
532             int x = (int) (in_buff[ix++] * (float)0x7FFFFFFF);
533             if (x < 0)
534                 x += 0x1000000;
535             out_buff[ox++] = (byte) (x >>> 16);
536             out_buff[ox++] = (byte) (x >>> 8);
537             out_buff[ox++] = (byte) x;
538         }
539         return out_buff;
540     }
541 }
542
543 // PCM 24 bit, unsigned, little-endian
544 private static class AudioFloatConversion24UL extends AudioFloatConverter {
545     public float[] toFloatArray(byte[] in_buff, int in_offset,
546         float[] out_buff, int out_offset, int out_len) {
547         int ix = in_offset;
548         int ox = out_offset;
549         for (int i = 0; i < out_len; i++) {
550             int x = (in_buff[ix++] & 0xFF) | ((in_buff[ix++] & 0xFF) << 8)
551                 | ((in_buff[ix++] & 0xFF) << 16);
552             x -= 0x7FFFFFFF;
553             out_buff[ox++] = x * (1.0f / (float)0x7FFFFFFF);
554         }
555         return out_buff;
556     }
557 }

```

```

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)0x7FFFFFFF);
564             x += 0x7FFFFFFF;
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 -= 0x7FFFFFFF;
583             out_buff[ox++] = x * (1.0f / (float)0x7FFFFFFF);
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)0x7FFFFFFF);
594             x += 0x7FFFFFFF;
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);

```

```

619         out_buff[ox++] = x * (1.0f / (float)0x7FFFFFFF);
620     }
621     return out_buff;
622 }
623
624 public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
625     byte[] out_buff, int out_offset) {
626     int ix = in_offset;
627     int ox = out_offset;
628     for (int i = 0; i < in_len; i++) {
629         int x = (int) (in_buff[ix++] * (float)0x7FFFFFFF);
630         out_buff[ox++] = (byte) x;
631         out_buff[ox++] = (byte) (x >>> 8);
632         out_buff[ox++] = (byte) (x >>> 16);
633         out_buff[ox++] = (byte) (x >>> 24);
634     }
635     return out_buff;
636 }
637
638
639 // PCM 32 bit, signed, big-endian
640 private static class AudioFloatConversion32SB extends AudioFloatConverter {
641     public float[] toFloatArray(byte[] in_buff, int in_offset,
642         float[] out_buff, int out_offset, int out_len) {
643         int ix = in_offset;
644         int ox = out_offset;
645         for (int i = 0; i < out_len; i++) {
646             int x = ((in_buff[ix++] & 0xFF) << 24) |
647                 ((in_buff[ix++] & 0xFF) << 16) |
648                 ((in_buff[ix++] & 0xFF) << 8) | (in_buff[ix++] & 0xFF);
649             out_buff[ox++] = x * (1.0f / (float)0x7FFFFFFF);
650         }
651         return out_buff;
652     }
653
654     public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
655         byte[] out_buff, int out_offset) {
656         int ix = in_offset;
657         int ox = out_offset;
658         for (int i = 0; i < in_len; i++) {
659             int x = (int) (in_buff[ix++] * (float)0x7FFFFFFF);
660             out_buff[ox++] = (byte) (x >>> 24);
661             out_buff[ox++] = (byte) (x >>> 16);
662             out_buff[ox++] = (byte) (x >>> 8);
663             out_buff[ox++] = (byte) x;
664         }
665         return out_buff;
666     }
667 }
668
669 // PCM 32 bit, unsigned, little-endian
670 private static class AudioFloatConversion32UL extends AudioFloatConverter {
671     public float[] toFloatArray(byte[] in_buff, int in_offset,
672         float[] out_buff, int out_offset, int out_len) {
673         int ix = in_offset;
674         int ox = out_offset;
675         for (int i = 0; i < out_len; i++) {
676             int x = (in_buff[ix++] & 0xFF) | ((in_buff[ix++] & 0xFF) << 8) |
677                 ((in_buff[ix++] & 0xFF) << 16) |
678                 ((in_buff[ix++] & 0xFF) << 24);
679             x -= 0x7FFFFFFF;
680             out_buff[ox++] = x * (1.0f / (float)0x7FFFFFFF);

```

```

681     }
682     return out_buff;
683 }
684
685 public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
686     byte[] out_buff, int out_offset) {
687     int ix = in_offset;
688     int ox = out_offset;
689     for (int i = 0; i < in_len; i++) {
690         int x = (int) (in_buff[ix++] * (float)0x7FFFFFFF);
691         x += 0x7FFFFFFF;
692         out_buff[ox++] = (byte) x;
693         out_buff[ox++] = (byte) (x >>> 8);
694         out_buff[ox++] = (byte) (x >>> 16);
695         out_buff[ox++] = (byte) (x >>> 24);
696     }
697     return out_buff;
698 }
699 }
700
701 // PCM 32 bit, unsigned, big-endian
702 private static class AudioFloatConversion32UB extends AudioFloatConverter {
703
704     public float[] toFloatArray(byte[] in_buff, int in_offset,
705         float[] out_buff, int out_offset, int out_len) {
706         int ix = in_offset;
707         int ox = out_offset;
708         for (int i = 0; i < out_len; i++) {
709             int x = ((in_buff[ix++] & 0xFF) << 24) |
710                 ((in_buff[ix++] & 0xFF) << 16) |
711                 ((in_buff[ix++] & 0xFF) << 8) | (in_buff[ix++] & 0xFF);
712             x -= 0x7FFFFFFF;
713             out_buff[ox++] = x * (1.0f / (float)0x7FFFFFFF);
714         }
715         return out_buff;
716     }
717
718     public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
719         byte[] out_buff, int out_offset) {
720         int ix = in_offset;
721         int ox = out_offset;
722         for (int i = 0; i < in_len; i++) {
723             int x = (int) (in_buff[ix++] * (float)0x7FFFFFFF);
724             x += 0x7FFFFFFF;
725             out_buff[ox++] = (byte) (x >>> 24);
726             out_buff[ox++] = (byte) (x >>> 16);
727             out_buff[ox++] = (byte) (x >>> 8);
728             out_buff[ox++] = (byte) x;
729         }
730         return out_buff;
731     }
732 }
733
734 /*****
735  *
736  * 32+ bit signed/unsigned, little/big-endian
737  *
738  *****/
739
740 // PCM 32+ bit, signed, little-endian
741 private static class AudioFloatConversion32xSL extends AudioFloatConverter {
742

```

```

743     final int xbytes;
744
745     public AudioFloatConversion32xSL(int xbytes) {
746         this.xbytes = xbytes;
747     }
748
749     public float[] toFloatArray(byte[] in_buff, int in_offset,
750                                float[] out_buff, int out_offset, int out_len) {
751         int ix = in_offset;
752         int ox = out_offset;
753         for (int i = 0; i < out_len; i++) {
754             ix += xbytes;
755             int x = (in_buff[ix++] & 0xFF) | ((in_buff[ix++] & 0xFF) << 8)
756                  | ((in_buff[ix++] & 0xFF) << 16)
757                  | ((in_buff[ix++] & 0xFF) << 24);
758             out_buff[ox++] = x * (1.0f / (float)0x7FFFFFFF);
759         }
760         return out_buff;
761     }
762
763     public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
764                              byte[] out_buff, int out_offset) {
765         int ix = in_offset;
766         int ox = out_offset;
767         for (int i = 0; i < in_len; i++) {
768             int x = (int) (in_buff[ix++] * (float)0x7FFFFFFF);
769             for (int j = 0; j < xbytes; j++) {
770                 out_buff[ox++] = 0;
771             }
772             out_buff[ox++] = (byte) x;
773             out_buff[ox++] = (byte) (x >>> 8);
774             out_buff[ox++] = (byte) (x >>> 16);
775             out_buff[ox++] = (byte) (x >>> 24);
776         }
777         return out_buff;
778     }
779 }
780
781 // PCM 32+ bit, signed, big-endian
782 private static class AudioFloatConversion32xSB extends AudioFloatConverter {
783
784     final int xbytes;
785
786     public AudioFloatConversion32xSB(int xbytes) {
787         this.xbytes = xbytes;
788     }
789
790     public float[] toFloatArray(byte[] in_buff, int in_offset,
791                                float[] out_buff, int out_offset, int out_len) {
792         int ix = in_offset;
793         int ox = out_offset;
794         for (int i = 0; i < out_len; i++) {
795             int x = ((in_buff[ix++] & 0xFF) << 24)
796                  | ((in_buff[ix++] & 0xFF) << 16)
797                  | ((in_buff[ix++] & 0xFF) << 8)
798                  | (in_buff[ix++] & 0xFF);
799             ix += xbytes;
800             out_buff[ox++] = x * (1.0f / (float)0x7FFFFFFF);
801         }
802         return out_buff;
803     }
804 }

```

```

805     public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
806         byte[] out_buff, int out_offset) {
807         int ix = in_offset;
808         int ox = out_offset;
809         for (int i = 0; i < in_len; i++) {
810             int x = (int) (in_buff[ix++] * (float)0x7FFFFFFF);
811             out_buff[ox++] = (byte) (x >>> 24);
812             out_buff[ox++] = (byte) (x >>> 16);
813             out_buff[ox++] = (byte) (x >>> 8);
814             out_buff[ox++] = (byte) x;
815             for (int j = 0; j < xbytes; j++) {
816                 out_buff[ox++] = 0;
817             }
818         }
819         return out_buff;
820     }
821 }
822
823 // PCM 32+ bit, unsigned, little-endian
824 private static class AudioFloatConversion32xUL extends AudioFloatConverter {
825
826     final int xbytes;
827
828     public AudioFloatConversion32xUL(int xbytes) {
829         this.xbytes = xbytes;
830     }
831
832     public float[] toFloatArray(byte[] in_buff, int in_offset,
833         float[] out_buff, int out_offset, int out_len) {
834         int ix = in_offset;
835         int ox = out_offset;
836         for (int i = 0; i < out_len; i++) {
837             ix += xbytes;
838             int x = (in_buff[ix++] & 0xFF) | ((in_buff[ix++] & 0xFF) << 8)
839                 | ((in_buff[ix++] & 0xFF) << 16)
840                 | ((in_buff[ix++] & 0xFF) << 24);
841             x -= 0x7FFFFFFF;
842             out_buff[ox++] = x * (1.0f / (float)0x7FFFFFFF);
843         }
844         return out_buff;
845     }
846
847     public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
848         byte[] out_buff, int out_offset) {
849         int ix = in_offset;
850         int ox = out_offset;
851         for (int i = 0; i < in_len; i++) {
852             int x = (int) (in_buff[ix++] * (float)0x7FFFFFFF);
853             x += 0x7FFFFFFF;
854             for (int j = 0; j < xbytes; j++) {
855                 out_buff[ox++] = 0;
856             }
857             out_buff[ox++] = (byte) x;
858             out_buff[ox++] = (byte) (x >>> 8);
859             out_buff[ox++] = (byte) (x >>> 16);
860             out_buff[ox++] = (byte) (x >>> 24);
861         }
862         return out_buff;
863     }
864 }
865
866 // 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) {

```

```

929         conv = new AudioFloatConversion32SB();
930     } else if (format.getSampleSizeInBits() > 32) {
931         conv = new AudioFloatConversion32xSB(((format
932             .getSampleSizeInBits() + 7) / 8) - 4);
933     }
934 } else {
935     if (format.getSampleSizeInBits() <= 8) {
936         conv = new AudioFloatConversion8S();
937     } else if (format.getSampleSizeInBits() > 8 &&
938         format.getSampleSizeInBits() <= 16) {
939         conv = new AudioFloatConversion16SL();
940     } else if (format.getSampleSizeInBits() > 16 &&
941         format.getSampleSizeInBits() <= 24) {
942         conv = new AudioFloatConversion24SL();
943     } else if (format.getSampleSizeInBits() > 24 &&
944         format.getSampleSizeInBits() <= 32) {
945         conv = new AudioFloatConversion32SL();
946     } else if (format.getSampleSizeInBits() > 32) {
947         conv = new AudioFloatConversion32xSL(((format
948             .getSampleSizeInBits() + 7) / 8) - 4);
949     }
950 }
951 } else if (format.getEncoding().equals(Encoding.PCM_UNSIGNED)) {
952     if (format.isBigEndian()) {
953         if (format.getSampleSizeInBits() <= 8) {
954             conv = new AudioFloatConversion8U();
955         } else if (format.getSampleSizeInBits() > 8 &&
956             format.getSampleSizeInBits() <= 16) {
957             conv = new AudioFloatConversion16UB();
958         } else if (format.getSampleSizeInBits() > 16 &&
959             format.getSampleSizeInBits() <= 24) {
960             conv = new AudioFloatConversion24UB();
961         } else if (format.getSampleSizeInBits() > 24 &&
962             format.getSampleSizeInBits() <= 32) {
963             conv = new AudioFloatConversion32UB();
964         } else if (format.getSampleSizeInBits() > 32) {
965             conv = new AudioFloatConversion32xUB(((
966                 format.getSampleSizeInBits() + 7) / 8) - 4);
967         }
968     } else {
969         if (format.getSampleSizeInBits() <= 8) {
970             conv = new AudioFloatConversion8U();
971         } else if (format.getSampleSizeInBits() > 8 &&
972             format.getSampleSizeInBits() <= 16) {
973             conv = new AudioFloatConversion16UL();
974         } else if (format.getSampleSizeInBits() > 16 &&
975             format.getSampleSizeInBits() <= 24) {
976             conv = new AudioFloatConversion24UL();
977         } else if (format.getSampleSizeInBits() > 24 &&
978             format.getSampleSizeInBits() <= 32) {
979             conv = new AudioFloatConversion32UL();
980         } else if (format.getSampleSizeInBits() > 32) {
981             conv = new AudioFloatConversion32xUL(((
982                 format.getSampleSizeInBits() + 7) / 8) - 4);
983         }
984     }
985 } else if (format.getEncoding().equals(PCM_FLOAT)) {
986     if (format.getSampleSizeInBits() == 32) {
987         if (format.isBigEndian())
988             conv = new AudioFloatConversion32B();
989         else
990             conv = new AudioFloatConversion32L();

```

```

991         } else if (format.getSampleSizeInBits() == 64) {
992             if (format.isBigEndian())
993                 conv = new AudioFloatConversion64B();
994             else
995                 conv = new AudioFloatConversion64L();
996         }
997     }
998 }
999
1000 if ((format.getEncoding().equals(Encoding.PCM_SIGNED) ||
1001     format.getEncoding().equals(Encoding.PCM_UNSIGNED)) &&
1002     (format.getSampleSizeInBits() % 8 != 0)) {
1003     conv = new AudioFloatLSBFilter(conv, format);
1004 }
1005
1006 if (conv != null)
1007     conv.format = format;
1008 return conv;
1009 }
1010
1011 private AudioFormat format;
1012
1013 public AudioFormat getFormat() {
1014     return format;
1015 }
1016
1017 public abstract float[] toFloatArray(byte[] in_buff, int in_offset,
1018     float[] out_buff, int out_offset, int out_len);
1019
1020 public float[] toFloatArray(byte[] in_buff, float[] out_buff,
1021     int out_offset, int out_len) {
1022     return toFloatArray(in_buff, 0, out_buff, out_offset, out_len);
1023 }
1024
1025 public float[] toFloatArray(byte[] in_buff, int in_offset,
1026     float[] out_buff, int out_len) {
1027     return toFloatArray(in_buff, in_offset, out_buff, 0, out_len);
1028 }
1029
1030 public float[] toFloatArray(byte[] in_buff, float[] out_buff, int out_len) {
1031     return toFloatArray(in_buff, 0, out_buff, 0, out_len);
1032 }
1033
1034 public float[] toFloatArray(byte[] in_buff, float[] out_buff) {
1035     return toFloatArray(in_buff, 0, out_buff, 0, out_buff.length);
1036 }
1037
1038 public abstract byte[] toByteArray(float[] in_buff, int in_offset,
1039     int in_len, byte[] out_buff, int out_offset);
1040
1041 public byte[] toByteArray(float[] in_buff, int in_len, byte[] out_buff,
1042     int out_offset) {
1043     return toByteArray(in_buff, 0, in_len, out_buff, out_offset);
1044 }
1045
1046 public byte[] toByteArray(float[] in_buff, int in_offset, int in_len,
1047     byte[] out_buff) {
1048     return toByteArray(in_buff, in_offset, in_len, out_buff, 0);
1049 }
1050
1051 public byte[] toByteArray(float[] in_buff, int in_len, byte[] out_buff) {
1052     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/little 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);
```

```

61         fsize = ((targetFormat.getSampleSizeInBits() + 7) / 8);
62     }
63
64     public int read() throws IOException {
65         byte[] b = new byte[1];
66         int ret = read(b);
67         if (ret < 0)
68             return ret;
69         return b[0] & 0xFF;
70     }
71
72     public int read(byte[] b, int off, int len) throws IOException {
73
74         int flen = len / fsize;
75         if (readfloatbuffer == null || readfloatbuffer.length < flen)
76             readfloatbuffer = new float[flen];
77         int ret = stream.read(readfloatbuffer, 0, flen);
78         if (ret < 0)
79             return ret;
80         converter.toByteArray(readfloatbuffer, 0, ret, b, off);
81         return ret * fsize;
82     }
83
84     public int available() throws IOException {
85         int ret = stream.available();
86         if (ret < 0)
87             return ret;
88         return ret * fsize;
89     }
90
91     public void close() throws IOException {
92         stream.close();
93     }
94
95     public synchronized void mark(int readlimit) {
96         stream.mark(readlimit * fsize);
97     }
98
99     public boolean markSupported() {
100         return stream.markSupported();
101     }
102
103     public synchronized void reset() throws IOException {
104         stream.reset();
105     }
106
107     public long skip(long n) throws IOException {
108         long ret = stream.skip(n / fsize);
109         if (ret < 0)
110             return ret;
111         return ret * fsize;
112     }
113
114 }
115
116 private static class AudioFloatInputStreamChannelMixer extends
117     AudioFloatInputStream {
118
119     private int targetChannels;
120
121     private int sourceChannels;
122

```

```

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             }
185         }
186     }

```

```

185         for (int c = 1; c < sourceChannels; c++) {
186             for (int i = c, ix = off; i < len2; i += cs, ix++) {
187                 b[ix] += conversion_buffer[i];
188             }
189         }
190         float vol = 1f / ((float) sourceChannels);
191         for (int i = 0, ix = off; i < len2; i += cs, ix++) {
192             b[ix] *= vol;
193         }
194     } else {
195         int minChannels = Math.min(sourceChannels, targetChannels);
196         int off_len = off + len;
197         int ct = targetChannels;
198         int cs = sourceChannels;
199         for (int c = 0; c < minChannels; c++) {
200             for (int i = off + c, ix = c; i < off_len; i += ct, ix += cs) {
201                 b[i] = conversion_buffer[ix];
202             }
203         }
204         for (int c = minChannels; c < targetChannels; c++) {
205             for (int i = off + c; i < off_len; i += ct) {
206                 b[i] = 0;
207             }
208         }
209     }
210     return (ret / sourceChannels) * targetChannels;
211 }
212
213 public void reset() throws IOException {
214     ais.reset();
215 }
216
217 public long skip(long len) throws IOException {
218     long ret = ais.skip((len / targetChannels) * sourceChannels);
219     if (ret < 0)
220         return ret;
221     return (ret / sourceChannels) * targetChannels;
222 }
223
224 }
225
226 private static class AudioFloatInputStreamResampler extends
227     AudioFloatInputStream {
228
229     private AudioFloatInputStream ais;
230
231     private AudioFormat targetFormat;
232
233     private float[] skipbuffer;
234
235     private SoftAbstractResampler resampler;
236
237     private float[] pitch = new float[1];
238
239     private float[] ibuffer2;
240
241     private float[][] ibuffer;
242
243     private float ibuffer_index = 0;
244
245     private int ibuffer_len = 0;
246

```



```

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     int ibuffer2_len = ibuffer2.length;
372     for (int c = 0; c < nrofchannels; c++) {
373         float[] buff = ibuffer[c];
374         for (int i = c, ix = pad2; i < ibuffer2_len; i += nrofchannels, ix++) {
375             buff[ix] = ibuffer2[i];
376         }
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 {

```

```

433     ais.reset();
434     if (mark_ibuffer == null)
435         return;
436     ibuffer_index = mark_ibuffer_index;
437     ibuffer_len = mark_ibuffer_len;
438     for (int c = 0; c < ibuffer.length; c++) {
439         float[] from = mark_ibuffer[c];
440         float[] to = ibuffer[c];
441         for (int i = 0; i < to.length; i++) {
442             to[i] = from[i];
443         }
444     }
445
446 }
447
448 public long skip(long len) throws IOException {
449     if (len < 0)
450         return 0;
451     if (skipbuffer == null)
452         skipbuffer = new float[1024 * targetFormat.getFrameSize()];
453     float[] l_skipbuffer = skipbuffer;
454     long remain = len;
455     while (remain > 0) {
456         int ret = read(l_skipbuffer, 0, (int) Math.min(remain,
457             skipbuffer.length));
458         if (ret < 0) {
459             if (remain == len)
460                 return ret;
461             break;
462         }
463         remain -= ret;
464     }
465     return len - remain;
466 }
467
468 }
469
470 private Encoding[] formats = { Encoding.PCM_SIGNED, Encoding.PCM_UNSIGNED,
471     AudioFloatConverter.PCM_FLOAT };
472
473 public AudioInputStream getAudioInputStream(Encoding targetEncoding,
474     AudioInputStream sourceStream) {
475     if (sourceStream.getFormat().getEncoding().equals(targetEncoding))
476         return sourceStream;
477     AudioFormat format = sourceStream.getFormat();
478     int channels = format.getChannels();
479     Encoding encoding = targetEncoding;
480     float samplerate = format.getSampleRate();
481     int bits = format.getSampleSizeInBits();
482     boolean bigendian = format.isBigEndian();
483     if (targetEncoding.equals(AudioFloatConverter.PCM_FLOAT))
484         bits = 32;
485     AudioFormat targetFormat = new AudioFormat(encoding, samplerate, bits,
486         channels, channels * bits / 8, samplerate, bigendian);
487     return getAudioInputStream(targetFormat, sourceStream);
488 }
489
490 public AudioInputStream getAudioInputStream(AudioFormat targetFormat,
491     AudioInputStream sourceStream) {
492     if (!isConversionSupported(targetFormat, sourceStream.getFormat()))
493         throw new IllegalArgumentException("Unsupported conversion:_"
494

```

```

495         + sourceStream.getFormat().toString() + " to "
496         + targetFormat.toString());
497     return getAudioInputStream(targetFormat, AudioFloatInputStream
498         .getInputStream(sourceStream));
499 }
500
501 public AudioInputStream getAudioInputStream(AudioFormat targetFormat,
502     AudioFloatInputStream sourceStream) {
503
504     if (!isConversionSupported(targetFormat, sourceStream.getFormat()))
505         throw new IllegalArgumentException("Unsupported conversion: "
506             + sourceStream.getFormat().toString() + " to "
507             + targetFormat.toString());
508     if (targetFormat.getChannels() != sourceStream.getFormat()
509         .getChannels())
510         sourceStream = new AudioFloatInputStreamChannelMixer(sourceStream,
511             targetFormat.getChannels());
512     if (Math.abs(targetFormat.getSampleRate()
513         - sourceStream.getFormat().getSampleRate()) > 0.000001)
514         sourceStream = new AudioFloatInputStreamResampler(sourceStream,
515             targetFormat);
516     return new AudioInputStream(new AudioFloatFormatConverterInputStream(
517         targetFormat, sourceStream), targetFormat, sourceStream
518         .getFrameLength());
519 }
520
521 public Encoding[] getSourceEncodings() {
522     return new Encoding[] { Encoding.PCM_SIGNED, Encoding.PCM_UNSIGNED,
523         AudioFloatConverter.PCM_FLOAT };
524 }
525
526 public Encoding[] getTargetEncodings() {
527     return new Encoding[] { Encoding.PCM_SIGNED, Encoding.PCM_UNSIGNED,
528         AudioFloatConverter.PCM_FLOAT };
529 }
530
531 public Encoding[] getTargetEncodings(AudioFormat sourceFormat) {
532     if (AudioFloatConverter.getConverter(sourceFormat) == null)
533         return new Encoding[0];
534     return new Encoding[] { Encoding.PCM_SIGNED, Encoding.PCM_UNSIGNED,
535         AudioFloatConverter.PCM_FLOAT };
536 }
537
538 public AudioFormat[] getTargetFormats(Encoding targetEncoding,
539     AudioFormat sourceFormat) {
540     if (AudioFloatConverter.getConverter(sourceFormat) == null)
541         return new AudioFormat[0];
542     int channels = sourceFormat.getChannels();
543
544     ArrayList<AudioFormat> formats = new ArrayList<AudioFormat>();
545
546     if (targetEncoding.equals(Encoding.PCM_SIGNED))
547         formats.add(new AudioFormat(Encoding.PCM_SIGNED,
548             AudioSystem.NOT_SPECIFIED, 8, channels, channels,
549             AudioSystem.NOT_SPECIFIED, false));
550     if (targetEncoding.equals(Encoding.PCM_UNSIGNED))
551         formats.add(new AudioFormat(Encoding.PCM_UNSIGNED,
552             AudioSystem.NOT_SPECIFIED, 8, channels, channels,
553             AudioSystem.NOT_SPECIFIED, false));
554
555     for (int bits = 16; bits < 32; bits += 8) {
556         if (targetEncoding.equals(Encoding.PCM_SIGNED)) {

```

```

557         formats.add(new AudioFormat(Encoding.PCM_SIGNED,
558             AudioSystem.NOT_SPECIFIED, bits, channels, channels
559             * bits / 8, AudioSystem.NOT_SPECIFIED, false));
560         formats.add(new AudioFormat(Encoding.PCM_SIGNED,
561             AudioSystem.NOT_SPECIFIED, bits, channels, channels
562             * bits / 8, AudioSystem.NOT_SPECIFIED, true));
563     }
564     if (targetEncoding.equals(Encoding.PCM_UNSIGNED)) {
565         formats.add(new AudioFormat(Encoding.PCM_UNSIGNED,
566             AudioSystem.NOT_SPECIFIED, bits, channels, channels
567             * bits / 8, AudioSystem.NOT_SPECIFIED, true));
568         formats.add(new AudioFormat(Encoding.PCM_UNSIGNED,
569             AudioSystem.NOT_SPECIFIED, bits, channels, channels
570             * bits / 8, AudioSystem.NOT_SPECIFIED, false));
571     }
572 }
573
574 if (targetEncoding.equals(AudioFloatConverter.PCM_FLOAT)) {
575     formats.add(new AudioFormat(AudioFloatConverter.PCM_FLOAT,
576         AudioSystem.NOT_SPECIFIED, 32, channels, channels * 4,
577         AudioSystem.NOT_SPECIFIED, false));
578     formats.add(new AudioFormat(AudioFloatConverter.PCM_FLOAT,
579         AudioSystem.NOT_SPECIFIED, 32, channels, channels * 4,
580         AudioSystem.NOT_SPECIFIED, true));
581     formats.add(new AudioFormat(AudioFloatConverter.PCM_FLOAT,
582         AudioSystem.NOT_SPECIFIED, 64, channels, channels * 8,
583         AudioSystem.NOT_SPECIFIED, false));
584     formats.add(new AudioFormat(AudioFloatConverter.PCM_FLOAT,
585         AudioSystem.NOT_SPECIFIED, 64, channels, channels * 8,
586         AudioSystem.NOT_SPECIFIED, true));
587 }
588
589 return formats.toArray(new AudioFormat[formats.size()]);
590 }
591
592 public boolean isConversionSupported(AudioFormat targetFormat,
593     AudioFormat sourceFormat) {
594     if (AudioFloatConverter.getConverter(sourceFormat) == null)
595         return false;
596     if (AudioFloatConverter.getConverter(targetFormat) == null)
597         return false;
598     if (sourceFormat.getChannels() <= 0)
599         return false;
600     if (targetFormat.getChannels() <= 0)
601         return false;
602     return true;
603 }
604
605 public boolean isConversionSupported(Encoding targetEncoding,
606     AudioFormat sourceFormat) {
607     if (AudioFloatConverter.getConverter(sourceFormat) == null)
608         return false;
609     for (int i = 0; i < formats.length; i++) {
610         if (targetEncoding.equals(formats[i]))
611             return true;
612     }
613     return false;
614 }
615
616 }

```

19 com/sun/media/sound/AudioFloatInputStream.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.io.ByteArrayInputStream;
28 import java.io.File;
29 import java.io.IOException;
30 import java.io.InputStream;
31 import java.net.URL;
32
33 import javax.sound.sampled.AudioFormat;
34 import javax.sound.sampled.AudioInputStream;
35 import javax.sound.sampled.AudioSystem;
36 import javax.sound.sampled.UnsupportedAudioFileException;
37
38 /**
39 * This class is used to create AudioFloatInputStream from AudioInputStream and
40 * byte buffers.
41 *
42 * @author Karl Helgason
43 */
44 public abstract class AudioFloatInputStream {
45
46     private static class ByteArrayAudioFloatInputStream
47         extends AudioFloatInputStream {
48
49         private int pos = 0;
50         private int markpos = 0;
51         private AudioFloatConverter converter;
52         private AudioFormat format;
53         private byte[] buffer;
54         private int buffer_offset;
55         private int buffer_len;
56         private int framesize_pc;
57
58         public ByteArrayAudioFloatInputStream(AudioFloatConverter converter,
59             byte[] buffer, int offset, int len) {
60             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 ByteArrayAudioFloatInputStream(converter, buffer,
243             offset, len);
244
245     InputStream stream = new ByteArrayInputStream(buffer, offset, len);
246     long aLen = format.getFrameSize() == AudioSystem.NOT_SPECIFIED

```

```

247         ? AudioSystem.NOT_SPECIFIED : len / format.getFrameSize();
248     AudioInputStream astream = new AudioInputStream(stream, format, aLen);
249     return getInputStream(astream);
250 }
251
252 public abstract AudioFormat getFormat();
253
254 public abstract long getFrameLength();
255
256 public abstract int read(float[] b, int off, int len) throws IOException;
257
258 public int read(float[] b) throws IOException {
259     return read(b, 0, b.length);
260 }
261
262 public float read() throws IOException {
263     float[] b = new float[1];
264     int ret = read(b, 0, 1);
265     if (ret == -1 || ret == 0)
266         return 0;
267     return b[0];
268 }
269
270 public abstract long skip(long len) throws IOException;
271
272 public abstract int available() throws IOException;
273
274 public abstract void close() throws IOException;
275
276 public abstract void mark(int readlimit);
277
278 public abstract boolean markSupported();
279
280 public abstract void reset() throws IOException;
281 }

```

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      *
```

```

61      * @param info a proposed list of tag/value pairs that will be sent on open.
62      * @return an array of AudioSynthesizerPropertyInfo objects
63      * describing possible properties. This array may be an empty array if
64      * no properties are required.
65      */
66      public AudioSynthesizerPropertyInfo[] getPropertyInfo(
67          Map<String, Object> info);
68
69      /**
70       * Opens the synthesizer and starts rendering audio into
71       * SourceDataLine.
72       *
73       * <p>An application opening a synthesizer explicitly with this call
74       * has to close the synthesizer by calling {@link #close}. This is
75       * necessary to release system resources and allow applications to
76       * exit cleanly.
77       *
78       * <p>Note that some synthesizers, once closed, cannot be reopened.
79       * Attempts to reopen such a synthesizer will always result in
80       * a MidiUnavailableException.
81       *
82       * @param line which AudioSynthesizer writes output audio into.
83       * If line is null, then line from system default mixer is used.
84       * @param info a Map<String, Object> object containing
85       * properties for additional configuration supported by synthesizer.
86       * If info is null then default settings are used.
87       *
88       * @throws MidiUnavailableException thrown if the synthesizer cannot be
89       * opened due to resource restrictions.
90       * @throws SecurityException thrown if the synthesizer cannot be
91       * opened due to security restrictions.
92       *
93       * @see #close
94       * @see #isOpen
95       */
96      public void open(SourceDataLine line, Map<String, Object> info)
97          throws MidiUnavailableException;
98
99      /**
100       * Opens the synthesizer and renders audio into returned
101       * AudioInputStream.
102       *
103       * <p>An application opening a synthesizer explicitly with this call
104       * has to close the synthesizer by calling {@link #close}. This is
105       * necessary to release system resources and allow applications to
106       * exit cleanly.
107       *
108       * <p>Note that some synthesizers, once closed, cannot be reopened.
109       * Attempts to reopen such a synthesizer will always result in
110       * a MidiUnavailableException.
111       *
112       * @param targetFormat specifies the AudioFormat
113       * used in returned AudioInputStream.
114       * @param info a Map<String, Object> object containing
115       * properties for additional configuration supported by synthesizer.
116       * If info is null then default settings are used.
117       *
118       * @throws MidiUnavailableException thrown if the synthesizer cannot be
119       * opened due to resource restrictions.
120       * @throws SecurityException thrown if the synthesizer cannot be
121       * opened due to security restrictions.
122       */

```

```
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

```
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 /**
28 * Information about property used in opening <code>AudioSynthesizer</code>.
29 *
30 * @author Karl Helgason
31 */
32 public class AudioSynthesizerPropertyInfo {
33
34     /**
35      * Constructs a <code>AudioSynthesizerPropertyInfo</code> object with a given
36      * name and value. The <code>description</code> and <code>choices</code>
37      * are initialized by <code>null</code> values.
38      *
39      * @param name the name of the property
40      * @param value the current value or class used for values.
41      *
42      */
43     public AudioSynthesizerPropertyInfo(String name, Object value) {
44         this.name = name;
45         if (value instanceof Class)
46             valueClass = (Class)value;
47         else
48         {
49             this.value = value;
50             if (value != null)
51                 valueClass = value.getClass();
52         }
53     }
54     /**
55      * The name of the property.
56      */
57     public String name;
58     /**
59      * A brief description of the property, which may be null.
60      */
61 }
```

```

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

```
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 /**
28 * This class is used to store information to describe soundbanks, instruments
29 * and samples. It is stored inside a "INFO" List Chunk inside DLS files.
30 *
31 * @author Karl Helgason
32 */
33 public class DLSInfo {
34
35     /**
36      * (INAM) Title or subject.
37      */
38     public String name = "untitled";
39     /**
40      * (ICRD) Date of creation, the format is: YYYY-MM-DD.
41      * For example 2007-01-01 for 1. january of year 2007.
42      */
43     public String creationDate = null;
44     /**
45      * (IENG) Name of engineer who created the object.
46      */
47     public String engineers = null;
48     /**
49      * (IPRD) Name of the product which the object is intended for.
50      */
51     public String product = null;
52     /**
53      * (ICOP) Copyright information.
54      */
55     public String copyright = null;
56     /**
57      * (ICMT) General comments. Doesn't contain newline characters.
58      */
59     public String comments = null;
60     /**
```

```

61     * (ISFT) Name of software package used to create the file.
62     */
63     public String tools = null;
64     /**
65     * (IARL) Where content is archived.
66     */
67     public String archival_location = null;
68     /**
69     * (IART) Artists of original content.
70     */
71     public String artist = null;
72     /**
73     * (ICMS) Names of persons or organizations who commissioned the file.
74     */
75     public String commissioned = null;
76     /**
77     * (IGNR) Genre of the work.
78     *     Example: jazz, classical, rock, etc.
79     */
80     public String genre = null;
81     /**
82     * (IKEY) List of keyword that describe the content.
83     *     Examples: FX, bird, piano, etc.
84     */
85     public String keywords = null;
86     /**
87     * (IMED) Describes original medium of the data.
88     *     For example: record, CD, etc.
89     */
90     public String medium = null;
91     /**
92     * (ISBJ) Description of the content.
93     */
94     public String subject = null;
95     /**
96     * (ISRC) Name of person or organization who supplied
97     *     original material for the file.
98     */
99     public String source = null;
100    /**
101    * (ISRF) Source media for sample data is from.
102    *     For example: CD, TV, etc.
103    */
104    public String source_form = null;
105    /**
106    * (ITCH) Technician who sample the file/object.
107    */
108    public String technician = null;
109 }

```

23 com/sun/media/sound/DLSInstrument.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.ArrayList;
28 import java.util.HashMap;
29 import java.util.List;
30 import java.util.Map;
31
32 import javax.sound.midi.Patch;
33
34 /**
35  * This class is used to store information to describe instrument.
36  * It contains list of regions and modulators.
37  * It is stored inside a "ins " List Chunk inside DLS files.
38  * In the DLS documentation a modulator is called articulator.
39  *
40  * @author Karl Helgason
41  */
42 public class DLSInstrument extends ModelInstrument {
43
44     protected int preset = 0;
45     protected int bank = 0;
46     protected boolean druminstrument = false;
47     protected byte[] guid = null;
48     protected DLSInfo info = new DLSInfo();
49     protected List<DLSRegion> regions = new ArrayList<DLSRegion>();
50     protected List<DLSModulator> modulators = new ArrayList<DLSModulator>();
51
52     public DLSInstrument() {
53         super(null, null, null, null);
54     }
55
56     public DLSInstrument(DLSSoundbank soundbank) {
57         super(soundbank, null, null, null);
58     }
59
60     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)

```

```

185         return ModelSource.SOURCE_NOTEON_KEYNUMBER;
186     if (src == DLSModulator.CONN_SRC_EG1)
187         return ModelSource.SOURCE_EG1;
188     if (src == DLSModulator.CONN_SRC_EG2)
189         return ModelSource.SOURCE_EG2;
190     if (src == DLSModulator.CONN_SRC_PITCHWHEEL)
191         return ModelSource.SOURCE_MIDI_PITCH;
192     if (src == DLSModulator.CONN_SRC_CC1)
193         return new ModelIdentifier("midi_cc", "1", 0);
194     if (src == DLSModulator.CONN_SRC_CC7)
195         return new ModelIdentifier("midi_cc", "7", 0);
196     if (src == DLSModulator.CONN_SRC_CC10)
197         return new ModelIdentifier("midi_cc", "10", 0);
198     if (src == DLSModulator.CONN_SRC_CC11)
199         return new ModelIdentifier("midi_cc", "11", 0);
200     if (src == DLSModulator.CONN_SRC_RPN0)
201         return new ModelIdentifier("midi_rpn", "0", 0);
202     if (src == DLSModulator.CONN_SRC_RPN1)
203         return new ModelIdentifier("midi_rpn", "1", 0);
204
205     if (src == DLSModulator.CONN_SRC_POLYPRESSURE)
206         return ModelSource.SOURCE_MIDI_POLY_PRESSURE;
207     if (src == DLSModulator.CONN_SRC_CHANNELPRESSURE)
208         return ModelSource.SOURCE_MIDI_CHANNEL_PRESSURE;
209     if (src == DLSModulator.CONN_SRC_VIBRATO)
210         return ModelSource.SOURCE_LFO2;
211     if (src == DLSModulator.CONN_SRC_MONOPRESSURE)
212         return ModelSource.SOURCE_MIDI_CHANNEL_PRESSURE;
213
214     if (src == DLSModulator.CONN_SRC_CC91)
215         return new ModelIdentifier("midi_cc", "91", 0);
216     if (src == DLSModulator.CONN_SRC_CC93)
217         return new ModelIdentifier("midi_cc", "93", 0);
218
219     return null;
220 }
221
222 private ModelConnectionBlock convertToModel(DLSModulator mod) {
223     ModelIdentifier source = convertToModelSrc(mod.getSource());
224     ModelIdentifier control = convertToModelSrc(mod.getControl());
225     ModelIdentifier destination_id =
226         convertToModelDest(mod.getDestination());
227
228     int scale = mod.getScale();
229     double f_scale;
230     if (scale == Integer.MIN_VALUE)
231         f_scale = Double.NEGATIVE_INFINITY;
232     else
233         f_scale = scale / 65536.0;
234
235     if (destination_id != null) {
236         ModelSource src = null;
237         ModelSource ctrl = null;
238         ModelConnectionBlock block = new ModelConnectionBlock();
239         if (control != null) {
240             ModelSource s = new ModelSource();
241             if (control == ModelSource.SOURCE_MIDI_PITCH) {
242                 ((ModelStandardTransform)s.getTransform()).setPolarity(
243                     ModelStandardTransform.POLARITY_BIPOLAR);
244             } else if (control == ModelSource.SOURCE_LFO1
245                 || control == ModelSource.SOURCE_LFO2) {
246                 ((ModelStandardTransform)s.getTransform()).setPolarity(

```

```

247         ModelStandardTransform.POLARITY_BIPOLAR);
248     }
249     s.setIdentifier(control);
250     block.addSource(s);
251     ctrl = s;
252 }
253 if (source != null) {
254     ModelSource s = new ModelSource();
255     if (source == ModelSource.SOURCE_MIDI_PITCH) {
256         ((ModelStandardTransform)s.getTransform()).setPolarity(
257             ModelStandardTransform.POLARITY_BIPOLAR);
258     } else if (source == ModelSource.SOURCE_LF01
259         || source == ModelSource.SOURCE_LF02) {
260         ((ModelStandardTransform)s.getTransform()).setPolarity(
261             ModelStandardTransform.POLARITY_BIPOLAR);
262     }
263     s.setIdentifier(source);
264     block.addSource(s);
265     src = s;
266 }
267 ModelDestination destination = new ModelDestination();
268 destination.setIdentifier(destination_id);
269 block.setDestination(destination);
270
271 if (mod.getVersion() == 1) {
272     //if (mod.getTransform() == DLSModulator.CONN_TRN_CONCAVE) {
273     //    ((ModelStandardTransform)destination.getTransform())
274     //        .setTransform(
275     //            ModelStandardTransform.TRANSFORM_CONCAVE);
276     //}
277     if (mod.getTransform() == DLSModulator.CONN_TRN_CONCAVE) {
278         if (src != null) {
279             ((ModelStandardTransform)src.getTransform())
280                 .setTransform(
281                     ModelStandardTransform.TRANSFORM_CONCAVE);
282             ((ModelStandardTransform)src.getTransform())
283                 .setDirection(
284                     ModelStandardTransform.DIRECTION_MAX2MIN);
285         }
286         if (ctrl != null) {
287             ((ModelStandardTransform)ctrl.getTransform())
288                 .setTransform(
289                     ModelStandardTransform.TRANSFORM_CONCAVE);
290             ((ModelStandardTransform)ctrl.getTransform())
291                 .setDirection(
292                     ModelStandardTransform.DIRECTION_MAX2MIN);
293         }
294     }
295
296 } else if (mod.getVersion() == 2) {
297     int transform = mod.getTransform();
298     int src_transform_invert = (transform >> 15) & 1;
299     int src_transform_bipolar = (transform >> 14) & 1;
300     int src_transform = (transform >> 10) & 8;
301     int ctr_transform_invert = (transform >> 9) & 1;
302     int ctr_transform_bipolar = (transform >> 8) & 1;
303     int ctr_transform = (transform >> 4) & 8;
304
305     if (src != null) {
306         int trans = ModelStandardTransform.TRANSFORM_LINEAR;
307         if (src_transform == DLSModulator.CONN_TRN_SWITCH)

```

```

309         trans = ModelStandardTransform.TRANSFORM_SWITCH;
310     if (src_transform == DLSModulator.CONN_TRN_CONCAVE)
311         trans = ModelStandardTransform.TRANSFORM_CONCAVE;
312     if (src_transform == DLSModulator.CONN_TRN_CONVEX)
313         trans = ModelStandardTransform.TRANSFORM_CONVEX;
314     ((ModelStandardTransform)src.getTransform())
315         .setTransform(trans);
316     ((ModelStandardTransform)src.getTransform())
317         .setPolarity(src_transform_bipolar == 1);
318     ((ModelStandardTransform)src.getTransform())
319         .setDirection(src_transform_invert == 1);
320
321 }
322
323 if (ctrl != null) {
324     int trans = ModelStandardTransform.TRANSFORM_LINEAR;
325     if (ctr_transform == DLSModulator.CONN_TRN_SWITCH)
326         trans = ModelStandardTransform.TRANSFORM_SWITCH;
327     if (ctr_transform == DLSModulator.CONN_TRN_CONCAVE)
328         trans = ModelStandardTransform.TRANSFORM_CONCAVE;
329     if (ctr_transform == DLSModulator.CONN_TRN_CONVEX)
330         trans = ModelStandardTransform.TRANSFORM_CONVEX;
331     ((ModelStandardTransform)ctrl.getTransform())
332         .setTransform(trans);
333     ((ModelStandardTransform)ctrl.getTransform())
334         .setPolarity(ctr_transform_bipolar == 1);
335     ((ModelStandardTransform)ctrl.getTransform())
336         .setDirection(ctr_transform_invert == 1);
337 }
338
339 /* No output transforms are defined the DLS Level 2
340 int out_transform = transform % 8;
341 int trans = ModelStandardTransform.TRANSFORM_LINEAR;
342 if (out_transform == DLSModulator.CONN_TRN_SWITCH)
343     trans = ModelStandardTransform.TRANSFORM_SWITCH;
344 if (out_transform == DLSModulator.CONN_TRN_CONCAVE)
345     trans = ModelStandardTransform.TRANSFORM_CONCAVE;
346 if (out_transform == DLSModulator.CONN_TRN_CONVEX)
347     trans = ModelStandardTransform.TRANSFORM_CONVEX;
348 if (ctrl != null) {
349     ((ModelStandardTransform)destination.getTransform())
350         .setTransform(trans);
351 }
352 */
353
354 }
355
356 block.setScale(f_scale);
357
358 return block;
359 }
360
361 return null;
362 }
363
364 public ModelPerformer[] getPerformers() {
365     List<ModelPerformer> performers = new ArrayList<ModelPerformer>();
366
367     Map<String, DLSModulator> modmap = new HashMap<String, DLSModulator>();
368     for (DLSModulator mod: getModulators()) {
369         modmap.put(mod.getSource() + "x" + mod.getControl() + "=" +
370             mod.getDestination(), mod);

```



```

371 }
372
373 Map<String, DLSModulator> insmodmap =
374     new HashMap<String, DLSModulator>();
375
376 for (DLSRegion zone: regions) {
377     ModelPerformer performer = new ModelPerformer();
378     performer.setName(zone.getSample().getName());
379     performer.setSelfNonExclusive((zone.getFusoptions() &
380         DLSRegion.OPTION_SELFNONEXCLUSIVE) != 0);
381     performer.setExclusiveClass(zone.getExclusiveClass());
382     performer.setKeyFrom(zone.getKeyfrom());
383     performer.setKeyTo(zone.getKeyto());
384     performer.setVelFrom(zone.getVelfrom());
385     performer.setVelTo(zone.getVelto());
386
387     insmodmap.clear();
388     insmodmap.putAll(modmap);
389     for (DLSModulator mod: zone.getModulators()) {
390         insmodmap.put(mod.getSource() + "x" + mod.getControl() + "=" +
391             mod.getDestination(), mod);
392     }
393
394     List<ModelConnectionBlock> blocks = performer.getConnectionBlocks();
395     for (DLSModulator mod: insmodmap.values()) {
396         ModelConnectionBlock p = convertToModel(mod);
397         if (p != null)
398             blocks.add(p);
399     }
400
401     DLSSample sample = zone.getSample();
402     DLSSampleOptions sampleopt = zone.getSampleoptions();
403     if (sampleopt == null)
404         sampleopt = sample.getSampleoptions();
405
406     ModelByteBuffer buff = sample.getDataBuffer();
407
408     float pitchcorrection = (-sampleopt.unitynote * 100) +
409         sampleopt.finetune;
410
411     ModelByteBufferWavetable osc = new ModelByteBufferWavetable(buff,
412         sample.getFormat(), pitchcorrection);
413     osc.setAttenuation(osc.getAttenuation() / 65536f);
414     if (sampleopt.getLoops().size() != 0) {
415         DLSSampleLoop loop = sampleopt.getLoops().get(0);
416         osc.setLoopStart((int)loop.getStart());
417         osc.setLoopLength((int)loop.getLength());
418         if (loop.getType() == DLSSampleLoop.LOOP_TYPE_FORWARD)
419             osc.setLoopType(ModelWavetable.LOOP_TYPE_FORWARD);
420         if (loop.getType() == DLSSampleLoop.LOOP_TYPE_RELEASE)
421             osc.setLoopType(ModelWavetable.LOOP_TYPE_RELEASE);
422         else
423             osc.setLoopType(ModelWavetable.LOOP_TYPE_FORWARD);
424     }
425
426     performer.getConnectionBlocks().add(
427         new ModelConnectionBlock(SoftFilter.FILTERTYPE_LP12,
428             new ModelDestination(
429                 new ModelIdentifier("filter", "type", 1)))));
430
431     performer.getOscillators().add(osc);
432

```

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

24 com/sun/media/sound/DLSModulator.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 /**
28 * This class is used to store modulator/articulation data.
29 * A modulator connects one synthesizer source to
30 * a destination. For example a note on velocity
31 * can be mapped to the gain of the synthesized voice.
32 * It is stored as a "art1" or "art2" chunk inside DLS files.
33 *
34 * @author Karl Helgason
35 */
36 public class DLSModulator {
37
38     // DLS1 Destinations
39     public static final int CONN_DST_NONE = 0x000; // 0
40     public static final int CONN_DST_GAIN = 0x001; // cB
41     public static final int CONN_DST_PITCH = 0x003; // cent
42     public static final int CONN_DST_PAN = 0x004; // 0.1%
43     public static final int CONN_DST_LFO_FREQUENCY = 0x104; // cent (default 5 Hz)
44     public static final int CONN_DST_LFO_STARTDELAY = 0x105; // timecent
45     public static final int CONN_DST_EG1_ATTACKTIME = 0x206; // timecent
46     public static final int CONN_DST_EG1_DECAYTIME = 0x207; // timecent
47     public static final int CONN_DST_EG1_RELEASETIME = 0x209; // timecent
48     public static final int CONN_DST_EG1_SUSTAINLEVEL = 0x20A; // 0.1%
49     public static final int CONN_DST_EG2_ATTACKTIME = 0x30A; // timecent
50     public static final int CONN_DST_EG2_DECAYTIME = 0x30B; // timecent
51     public static final int CONN_DST_EG2_RELEASETIME = 0x30D; // timecent
52     public static final int CONN_DST_EG2_SUSTAINLEVEL = 0x30E; // 0.1%
53     // DLS2 Destinations
54     public static final int CONN_DST_KEYNUMBER = 0x005;
55     public static final int CONN_DST_LEFT = 0x010; // 0.1%
56     public static final int CONN_DST_RIGHT = 0x011; // 0.1%
57     public static final int CONN_DST_CENTER = 0x012; // 0.1%
58     public static final int CONN_DST_LEFTREAR = 0x013; // 0.1%
59     public static final int CONN_DST_RIGHTREAR = 0x014; // 0.1%
60     public static final int CONN_DST_LFE_CHANNEL = 0x015; // 0.1%
```

```

61 public static final int CONN_DST_CHORUS = 0x080; // 0.1%
62 public static final int CONN_DST_REVERB = 0x081; // 0.1%
63 public static final int CONN_DST_VIB_FREQUENCY = 0x114; // cent
64 public static final int CONN_DST_VIB_STARTDELAY = 0x115; // dB
65 public static final int CONN_DST_EG1_DELAYTIME = 0x20B; // timecent
66 public static final int CONN_DST_EG1_HOLDTIME = 0x20C; // timecent
67 public static final int CONN_DST_EG1_SHUTDOWNTIME = 0x20D; // timecent
68 public static final int CONN_DST_EG2_DELAYTIME = 0x30F; // timecent
69 public static final int CONN_DST_EG2_HOLDTIME = 0x310; // timecent
70 public static final int CONN_DST_FILTER_CUTOFF = 0x500; // cent
71 public static final int CONN_DST_FILTER_Q = 0x501; // dB
72
73 // DLS1 Sources
74 public static final int CONN_SRC_NONE = 0x000; // 1
75 public static final int CONN_SRC_LFO = 0x001; // linear (sine wave)
76 public static final int CONN_SRC_KEYONVELOCITY = 0x002; // ??db or velocity??
77 public static final int CONN_SRC_KEYNUMBER = 0x003; // ??cent or keynumber??
78 public static final int CONN_SRC_EG1 = 0x004; // linear direct from eg
79 public static final int CONN_SRC_EG2 = 0x005; // linear direct from eg
80 public static final int CONN_SRC_PITCHWHEEL = 0x006; // linear -1..1
81 public static final int CONN_SRC_CC1 = 0x081; // linear 0..1
82 public static final int CONN_SRC_CC7 = 0x087; // linear 0..1
83 public static final int CONN_SRC_CC10 = 0x08A; // linear 0..1
84 public static final int CONN_SRC_CC11 = 0x08B; // linear 0..1
85 public static final int CONN_SRC_RPN0 = 0x100; // ?? // Pitch Bend Range
86 public static final int CONN_SRC_RPN1 = 0x101; // ?? // Fine Tune
87 public static final int CONN_SRC_RPN2 = 0x102; // ?? // Course Tune
88 // DLS2 Sources
89 public static final int CONN_SRC_POLYPRESSURE = 0x007; // linear 0..1
90 public static final int CONN_SRC_CHANNELPRESSURE = 0x008; // linear 0..1
91 public static final int CONN_SRC_VIBRATO = 0x009; // linear 0..1
92 public static final int CONN_SRC_MONOPRESSURE = 0x00A; // linear 0..1
93 public static final int CONN_SRC_CC91 = 0x0DB; // linear 0..1
94 public static final int CONN_SRC_CC93 = 0x0DD; // linear 0..1
95 // DLS1 Transforms
96 public static final int CONN_TRN_NONE = 0x000;
97 public static final int CONN_TRN_CONCAVE = 0x001;
98 // DLS2 Transforms
99 public static final int CONN_TRN_CONVEX = 0x002;
100 public static final int CONN_TRN_SWITCH = 0x003;
101 public static final int DST_FORMAT_CB = 1;
102 public static final int DST_FORMAT_CENT = 1;
103 public static final int DST_FORMAT_TIMECENT = 2;
104 public static final int DST_FORMAT_PERCENT = 3;
105 protected int source;
106 protected int control;
107 protected int destination;
108 protected int transform;
109 protected int scale;
110 protected int version = 1;
111
112 public int getControl() {
113     return control;
114 }
115
116 public void setControl(int control) {
117     this.control = control;
118 }
119
120 public static int getDestinationFormat(int destination) {
121
122     if (destination == CONN_DST_GAIN)

```

```

123     return DST_FORMAT_CB;
124 if (destination == CONN_DST_PITCH)
125     return DST_FORMAT_CENT;
126 if (destination == CONN_DST_PAN)
127     return DST_FORMAT_PERCENT;
128
129 if (destination == CONN_DST_LFO_FREQUENCY)
130     return DST_FORMAT_CENT;
131 if (destination == CONN_DST_LFO_STARTDELAY)
132     return DST_FORMAT_TIMECENT;
133
134 if (destination == CONN_DST_EG1_ATTACKTIME)
135     return DST_FORMAT_TIMECENT;
136 if (destination == CONN_DST_EG1_DECAYTIME)
137     return DST_FORMAT_TIMECENT;
138 if (destination == CONN_DST_EG1_RELEASETIME)
139     return DST_FORMAT_TIMECENT;
140 if (destination == CONN_DST_EG1_SUSTAINLEVEL)
141     return DST_FORMAT_PERCENT;
142
143 if (destination == CONN_DST_EG2_ATTACKTIME)
144     return DST_FORMAT_TIMECENT;
145 if (destination == CONN_DST_EG2_DECAYTIME)
146     return DST_FORMAT_TIMECENT;
147 if (destination == CONN_DST_EG2_RELEASETIME)
148     return DST_FORMAT_TIMECENT;
149 if (destination == CONN_DST_EG2_SUSTAINLEVEL)
150     return DST_FORMAT_PERCENT;
151
152 if (destination == CONN_DST_KEYNUMBER)
153     return DST_FORMAT_CENT; // NOT SURE WITHOUT DLS 2 SPEC
154 if (destination == CONN_DST_LEFT)
155     return DST_FORMAT_CB;
156 if (destination == CONN_DST_RIGHT)
157     return DST_FORMAT_CB;
158 if (destination == CONN_DST_CENTER)
159     return DST_FORMAT_CB;
160 if (destination == CONN_DST_LEFTREAR)
161     return DST_FORMAT_CB;
162 if (destination == CONN_DST_RIGHTREAR)
163     return DST_FORMAT_CB;
164 if (destination == CONN_DST_LFE_CHANNEL)
165     return DST_FORMAT_CB;
166 if (destination == CONN_DST_CHORUS)
167     return DST_FORMAT_PERCENT;
168 if (destination == CONN_DST_REVERB)
169     return DST_FORMAT_PERCENT;
170
171 if (destination == CONN_DST_VIB_FREQUENCY)
172     return DST_FORMAT_CENT;
173 if (destination == CONN_DST_VIB_STARTDELAY)
174     return DST_FORMAT_TIMECENT;
175
176 if (destination == CONN_DST_EG1_DELAYTIME)
177     return DST_FORMAT_TIMECENT;
178 if (destination == CONN_DST_EG1_HOLDTIME)
179     return DST_FORMAT_TIMECENT;
180 if (destination == CONN_DST_EG1_SHUTDOWNTIME)
181     return DST_FORMAT_TIMECENT;
182
183 if (destination == CONN_DST_EG2_DELAYTIME)
184     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

```

```

247 if (destination == CONN_DST_VIB_FREQUENCY)
248     return "vib.freq";
249 if (destination == CONN_DST_VIB_STARTDELAY)
250     return "vib.delay";
251
252 if (destination == CONN_DST_EG1_DELAYTIME)
253     return "eg1.delay";
254 if (destination == CONN_DST_EG1_HOLDTIME)
255     return "eg1.hold";
256 if (destination == CONN_DST_EG1_SHUTDOWNTIME)
257     return "eg1.shutdown";
258
259 if (destination == CONN_DST_EG2_DELAYTIME)
260     return "eg2.delay";
261 if (destination == CONN_DST_EG2_HOLDTIME)
262     return "eg.2hold";
263
264 if (destination == CONN_DST_FILTER_CUTOFF)
265     return "filter.cutoff"; // NOT SURE WITHOUT DLS 2 SPEC
266 if (destination == CONN_DST_FILTER_Q)
267     return "filter.q"; // NOT SURE WITHOUT DLS 2 SPEC
268
269 return null;
270 }
271
272 public static String getSourceName(int source) {
273
274     if (source == CONN_SRC_NONE)
275         return "none";
276     if (source == CONN_SRC_LFO)
277         return "lfo";
278     if (source == CONN_SRC_KEYONVELOCITY)
279         return "keyonvelocity";
280     if (source == CONN_SRC_KEYNUMBER)
281         return "keynumber";
282     if (source == CONN_SRC_EG1)
283         return "eg1";
284     if (source == CONN_SRC_EG2)
285         return "eg2";
286     if (source == CONN_SRC_PITCHWHEEL)
287         return "pitchwheel";
288     if (source == CONN_SRC_CC1)
289         return "cc1";
290     if (source == CONN_SRC_CC7)
291         return "cc7";
292     if (source == CONN_SRC_CC10)
293         return "c10";
294     if (source == CONN_SRC_CC11)
295         return "cc11";
296
297     if (source == CONN_SRC_POLYPRESSURE)
298         return "polypressure";
299     if (source == CONN_SRC_CHANNELPRESSURE)
300         return "channelpressure";
301     if (source == CONN_SRC_VIBRATO)
302         return "vibrato";
303     if (source == CONN_SRC_MONOPRESSURE)
304         return "monopressure";
305     if (source == CONN_SRC_CC91)
306         return "cc91";
307     if (source == CONN_SRC_CC93)
308         return "cc93";

```

```
309         return null;
310     }
311
312     public int getDestination() {
313         return destination;
314     }
315
316     public void setDestination(int destination) {
317         this.destination = destination;
318     }
319
320     public int getScale() {
321         return scale;
322     }
323
324     public void setScale(int scale) {
325         this.scale = scale;
326     }
327
328     public int getSource() {
329         return source;
330     }
331
332     public void setSource(int source) {
333         this.source = source;
334     }
335
336     public int getVersion() {
337         return version;
338     }
339
340     public void setVersion(int version) {
341         this.version = version;
342     }
343
344     public int getTransform() {
345         return transform;
346     }
347
348     public void setTransform(int transform) {
349         this.transform = transform;
350     }
351 }
```


25 com/sun/media/sound/DLSRegion.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.ArrayList;
28 import java.util.List;
29
30 /**
31 * This class is used to store region parts for instrument.
32 * A region has a velocity and key range which it response to.
33 * And it has a list of modulators/articulators which
34 * is used how to synthesize a single voice.
35 * It is stored inside a "rgn " List Chunk inside DLS files.
36 *
37 * @author Karl Helgason
38 */
39 public class DLSRegion {
40
41     public final static int OPTION_SELFNONEXCLUSIVE = 0x0001;
42     protected List<DLSModulator> modulators = new ArrayList<DLSModulator>();
43     protected int keyfrom;
44     protected int keyto;
45     protected int velfrom;
46     protected int velto;
47     protected int options;
48     protected int exclusiveClass;
49     protected int fusoptions;
50     protected int phasegroup;
51     protected long channel;
52     protected DLSSample sample = null;
53     protected DLSSampleOptions sampleoptions;
54
55     public List<DLSModulator> getModulators() {
56         return modulators;
57     }
58
59     public long getChannel() {
60         return channel;
```

```

61     }
62
63     public void setChannel(long channel) {
64         this.channel = channel;
65     }
66
67     public int getExclusiveClass() {
68         return exclusiveClass;
69     }
70
71     public void setExclusiveClass(int exclusiveClass) {
72         this.exclusiveClass = exclusiveClass;
73     }
74
75     public int getFusoptions() {
76         return fusoptions;
77     }
78
79     public void setFusoptions(int fusoptions) {
80         this.fusoptions = fusoptions;
81     }
82
83     public int getKeyfrom() {
84         return keyfrom;
85     }
86
87     public void setKeyfrom(int keyfrom) {
88         this.keyfrom = keyfrom;
89     }
90
91     public int getKeyto() {
92         return keyto;
93     }
94
95     public void setKeyto(int keyto) {
96         this.keyto = keyto;
97     }
98
99     public int getOptions() {
100         return options;
101     }
102
103     public void setOptions(int options) {
104         this.options = options;
105     }
106
107     public int getPhasegroup() {
108         return phasegroup;
109     }
110
111     public void setPhasegroup(int phasegroup) {
112         this.phasegroup = phasegroup;
113     }
114
115     public DLSSample getSample() {
116         return sample;
117     }
118
119     public void setSample(DLSSample sample) {
120         this.sample = sample;
121     }
122

```

```

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<DLSSModulator> 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

```
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.io.InputStream;
28 import javax.sound.midi.Soundbank;
29 import javax.sound.midi.SoundbankResource;
30 import javax.sound.sampled.AudioFormat;
31 import javax.sound.sampled.AudioInputStream;
32
33 /**
34 * This class is used to store the sample data itself.
35 * A sample is encoded as PCM audio stream
36 * and in DLS Level 1 files it is always a mono 8/16 bit stream.
37 * They are stored just like RIFF WAVE files are stored.
38 * It is stored inside a "wave" List Chunk inside DLS files.
39 *
40 * @author Karl Helgason
41 */
42 public class DLSSample extends SoundbankResource {
43
44     protected byte[] guid = null;
45     protected DLSInfo info = new DLSInfo();
46     protected DLSSampleOptions sampleoptions;
47     protected ModelByteBuffer data;
48     protected AudioFormat format;
49
50     public DLSSample(Soundbank soundBank) {
51         super(soundBank, null, AudioInputStream.class);
52     }
53
54     public DLSSample() {
55         super(null, null, AudioInputStream.class);
56     }
57
58     public DLSInfo getInfo() {
59         return info;
60     }
61 }
```

```

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

```
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 /**
28 * This class is used to store loop points inside DLSSampleOptions class.
29 *
30 * @author Karl Helgason
31 */
32 public class DLSSampleLoop {
33
34     public final static int LOOP_TYPE_FORWARD = 0;
35     public final static int LOOP_TYPE_RELEASE = 1;
36     protected long type;
37     protected long start;
38     protected long length;
39
40     public long getLength() {
41         return length;
42     }
43
44     public void setLength(long length) {
45         this.length = length;
46     }
47
48     public long getStart() {
49         return start;
50     }
51
52     public void setStart(long start) {
53         this.start = start;
54     }
55
56     public long getType() {
57         return type;
58     }
59
60     public void setType(long type) {
```

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

28 com/sun/media/sound/DLSSampleOptions.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.ArrayList;
28 import java.util.List;
29
30 /**
31 * This class stores options how to playback sampled data like pitch/tuning,
32 * attenuation and loops.
33 * It is stored as a "wsmp" chunk inside DLS files.
34 *
35 * @author Karl Helgason
36 */
37 public class DLSSampleOptions {
38
39     protected int unitynote;
40     protected short finetune;
41     protected int attenuation;
42     protected long options;
43     protected List<DLSSampleLoop> loops = new ArrayList<DLSSampleLoop>();
44
45     public int getAttenuation() {
46         return attenuation;
47     }
48
49     public void setAttenuation(int attenuation) {
50         this.attenuation = attenuation;
51     }
52
53     public short getFinetune() {
54         return finetune;
55     }
56
57     public void setFinetune(short finetune) {
58         this.finetune = finetune;
59     }
60 }
```



```
61     public List<DLSSampleLoop> getLoops() {
62         return loops;
63     }
64
65     public long getOptions() {
66         return options;
67     }
68
69     public void setOptions(long options) {
70         this.options = options;
71     }
72
73     public int getUnitynote() {
74         return unitynote;
75     }
76
77     public void setUnitynote(int unitynote) {
78         this.unitynote = unitynote;
79     }
80 }
```

29 com/sun/media/sound/DLSSoundbank.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.io.File;
28 import java.io.FileInputStream;
29 import java.io.IOException;
30 import java.io.InputStream;
31 import java.io.OutputStream;
32 import java.net.URL;
33 import java.util.ArrayList;
34 import java.util.Arrays;
35 import java.util.HashMap;
36 import java.util.List;
37 import java.util.Map;
38 import java.util.Stack;
39
40 import javax.sound.midi.Instrument;
41 import javax.sound.midi.Patch;
42 import javax.sound.midi.Soundbank;
43 import javax.sound.midi.SoundbankResource;
44 import javax.sound.sampled.AudioFormat;
45 import javax.sound.sampled.AudioInputStream;
46 import javax.sound.sampled.AudioSystem;
47 import javax.sound.sampled.AudioFormat.Encoding;
48
49 /**
50 * A DLS Level 1 and Level 2 soundbank reader (from files/url/streams).
51 *
52 * @author Karl Helgason
53 */
54 public class DLSSoundbank implements Soundbank {
55
56     static private class DLSID {
57         long i1;
58         int s1;
59         int s2;
60         int x1;
```

```

61     int x2;
62     int x3;
63     int x4;
64     int x5;
65     int x6;
66     int x7;
67     int x8;
68
69     private DLSID() {
70     }
71
72     public DLSID(long i1, int s1, int s2, int x1, int x2, int x3, int x4,
73         int x5, int x6, int x7, int x8) {
74         this.i1 = i1;
75         this.s1 = s1;
76         this.s2 = s2;
77         this.x1 = x1;
78         this.x2 = x2;
79         this.x3 = x3;
80         this.x4 = x4;
81         this.x5 = x5;
82         this.x6 = x6;
83         this.x7 = x7;
84         this.x8 = x8;
85     }
86
87     public static DLSID read(RIFFReader riff) throws IOException {
88         DLSID d = new DLSID();
89         d.i1 = riff.readUnsignedInt();
90         d.s1 = riff.readUnsignedShort();
91         d.s2 = riff.readUnsignedShort();
92         d.x1 = riff.readUnsignedByte();
93         d.x2 = riff.readUnsignedByte();
94         d.x3 = riff.readUnsignedByte();
95         d.x4 = riff.readUnsignedByte();
96         d.x5 = riff.readUnsignedByte();
97         d.x6 = riff.readUnsignedByte();
98         d.x7 = riff.readUnsignedByte();
99         d.x8 = riff.readUnsignedByte();
100        return d;
101    }
102
103    public int hashCode() {
104        return (int)i1;
105    }
106
107    public boolean equals(Object obj) {
108        if (!(obj instanceof DLSID)) {
109            return false;
110        }
111        DLSID t = (DLSID) obj;
112        return i1 == t.i1 && s1 == t.s1 && s2 == t.s2
113            && x1 == t.x1 && x2 == t.x2 && x3 == t.x3 && x4 == t.x4
114            && x5 == t.x5 && x6 == t.x6 && x7 == t.x7 && x8 == t.x8;
115    }
116 }
117
118 /** X = X & Y */
119 private static final int DLS_CDL_AND = 0x0001;
120 /** X = X | Y */
121 private static final int DLS_CDL_OR = 0x0002;
122 /** X = X ^ Y */

```

```

123 private static final int DLS_CDL_XOR = 0x0003;
124 /** X = X + Y */
125 private static final int DLS_CDL_ADD = 0x0004;
126 /** X = X - Y */
127 private static final int DLS_CDL_SUBTRACT = 0x0005;
128 /** X = X * Y */
129 private static final int DLS_CDL_MULTIPLY = 0x0006;
130 /** X = X / Y */
131 private static final int DLS_CDL_DIVIDE = 0x0007;
132 /** X = X && Y */
133 private static final int DLS_CDL_LOGICAL_AND = 0x0008;
134 /** X = X || Y */
135 private static final int DLS_CDL_LOGICAL_OR = 0x0009;
136 /** X = (X < Y) */
137 private static final int DLS_CDL_LT = 0x000A;
138 /** X = (X <= Y) */
139 private static final int DLS_CDL_LE = 0x000B;
140 /** X = (X > Y) */
141 private static final int DLS_CDL_GT = 0x000C;
142 /** X = (X >= Y) */
143 private static final int DLS_CDL_GE = 0x000D;
144 /** X = (X == Y) */
145 private static final int DLS_CDL_EQ = 0x000E;
146 /** X = !X */
147 private static final int DLS_CDL_NOT = 0x000F;
148 /** 32-bit constant */
149 private static final int DLS_CDL_CONST = 0x0010;
150 /** 32-bit value returned from query */
151 private static final int DLS_CDL_QUERY = 0x0011;
152 /** 32-bit value returned from query */
153 private static final int DLS_CDL_QUERY_SUPPORTED = 0x0012;
154
155 private static final DLSID DLSID_GMinHardware = new DLSID(0x178f2f24,
156 0xc364, 0x11d1, 0xa7, 0x60, 0x00, 0x00, 0xf8, 0x75, 0xac, 0x12);
157 private static final DLSID DLSID_GSInHardware = new DLSID(0x178f2f25,
158 0xc364, 0x11d1, 0xa7, 0x60, 0x00, 0x00, 0xf8, 0x75, 0xac, 0x12);
159 private static final DLSID DLSID_XGInHardware = new DLSID(0x178f2f26,
160 0xc364, 0x11d1, 0xa7, 0x60, 0x00, 0x00, 0xf8, 0x75, 0xac, 0x12);
161 private static final DLSID DLSID_SupportsDLS1 = new DLSID(0x178f2f27,
162 0xc364, 0x11d1, 0xa7, 0x60, 0x00, 0x00, 0xf8, 0x75, 0xac, 0x12);
163 private static final DLSID DLSID_SupportsDLS2 = new DLSID(0xf14599e5,
164 0x4689, 0x11d2, 0xaf, 0xa6, 0x0, 0xaa, 0x0, 0x24, 0xd8, 0xb6);
165 private static final DLSID DLSID_SampleMemorySize = new DLSID(0x178f2f28,
166 0xc364, 0x11d1, 0xa7, 0x60, 0x00, 0x00, 0xf8, 0x75, 0xac, 0x12);
167 private static final DLSID DLSID_ManufacturersID = new DLSID(0xb03e1181,
168 0x8095, 0x11d2, 0xa1, 0xef, 0x0, 0x60, 0x8, 0x33, 0xdb, 0xd8);
169 private static final DLSID DLSID_ProductID = new DLSID(0xb03e1182,
170 0x8095, 0x11d2, 0xa1, 0xef, 0x0, 0x60, 0x8, 0x33, 0xdb, 0xd8);
171 private static final DLSID DLSID_SamplePlaybackRate = new DLSID(0x2a91f713,
172 0xa4bf, 0x11d2, 0xbb, 0xdf, 0x0, 0x60, 0x8, 0x33, 0xdb, 0xd8);
173
174 private long major = -1;
175 private long minor = -1;
176
177 private DLSInfo info = new DLSInfo();
178
179 private List<DLSInstrument> instruments = new ArrayList<DLSInstrument>();
180 private List<DLSSample> samples = new ArrayList<DLSSample>();
181
182 private boolean largeFormat = false;
183 private File sampleFile;
184

```

```

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             }

```

```

247         if (chunk.getFormat().equals("vers")) {
248             major = chunk.readUnsignedInt();
249             minor = chunk.readUnsignedInt();
250         }
251     }
252 }
253
254 for (Map.Entry<DLSRegion, Long> entry : temp_rgnassign.entrySet()) {
255     entry.getKey().sample = samples.get((int)entry.getValue().longValue());
256 }
257
258 temp_rgnassign = null;
259 }
260
261 private boolean cdlIsQuerySupported(DLSID uuid) {
262     return uuid.equals(DLSID_GMInHardware)
263         || uuid.equals(DLSID_GSInHardware)
264         || uuid.equals(DLSID_XGInHardware)
265         || uuid.equals(DLSID_SupportsDLS1)
266         || uuid.equals(DLSID_SupportsDLS2)
267         || uuid.equals(DLSID_SampleMemorySize)
268         || uuid.equals(DLSID_ManufacturersID)
269         || uuid.equals(DLSID_ProductID)
270         || uuid.equals(DLSID_SamplePlaybackRate);
271 }
272
273 private long cdlQuery(DLSID uuid) {
274     if (uuid.equals(DLSID_GMInHardware))
275         return 1;
276     if (uuid.equals(DLSID_GSInHardware))
277         return 0;
278     if (uuid.equals(DLSID_XGInHardware))
279         return 0;
280     if (uuid.equals(DLSID_SupportsDLS1))
281         return 1;
282     if (uuid.equals(DLSID_SupportsDLS2))
283         return 1;
284     if (uuid.equals(DLSID_SampleMemorySize))
285         return Runtime.getRuntime().totalMemory();
286     if (uuid.equals(DLSID_ManufacturersID))
287         return 0;
288     if (uuid.equals(DLSID_ProductID))
289         return 0;
290     if (uuid.equals(DLSID_SamplePlaybackRate))
291         return 44100;
292     return 0;
293 }
294
295
296 // Reading cdl-ck Chunk
297 // "cdl " chunk can only appear inside : DLS,lart,lar2,rgn,rgn2
298 private boolean readCdlChunk(RIFFReader riff) throws IOException {
299
300     DLSID uuid;
301     long x;
302     long y;
303     Stack<Long> stack = new Stack<Long>();
304
305     while (riff.available() != 0) {
306         int opcode = riff.readUnsignedShort();
307         switch (opcode) {
308             case DLS_CDL_AND:

```

```

309     x = stack.pop();
310     y = stack.pop();
311     stack.push(Long.valueOf(((x != 0) && (y != 0)) ? 1 : 0));
312     break;
313 case DLS_CDL_OR:
314     x = stack.pop();
315     y = stack.pop();
316     stack.push(Long.valueOf(((x != 0) || (y != 0)) ? 1 : 0));
317     break;
318 case DLS_CDL_XOR:
319     x = stack.pop();
320     y = stack.pop();
321     stack.push(Long.valueOf(((x != 0) ^ (y != 0)) ? 1 : 0));
322     break;
323 case DLS_CDL_ADD:
324     x = stack.pop();
325     y = stack.pop();
326     stack.push(Long.valueOf(x + y));
327     break;
328 case DLS_CDL_SUBTRACT:
329     x = stack.pop();
330     y = stack.pop();
331     stack.push(Long.valueOf(x - y));
332     break;
333 case DLS_CDL_MULTIPLY:
334     x = stack.pop();
335     y = stack.pop();
336     stack.push(Long.valueOf(x * y));
337     break;
338 case DLS_CDL_DIVIDE:
339     x = stack.pop();
340     y = stack.pop();
341     stack.push(Long.valueOf(x / y));
342     break;
343 case DLS_CDL_LOGICAL_AND:
344     x = stack.pop();
345     y = stack.pop();
346     stack.push(Long.valueOf(((x != 0) && (y != 0)) ? 1 : 0));
347     break;
348 case DLS_CDL_LOGICAL_OR:
349     x = stack.pop();
350     y = stack.pop();
351     stack.push(Long.valueOf(((x != 0) || (y != 0)) ? 1 : 0));
352     break;
353 case DLS_CDL_LT:
354     x = stack.pop();
355     y = stack.pop();
356     stack.push(Long.valueOf((x < y) ? 1 : 0));
357     break;
358 case DLS_CDL_LE:
359     x = stack.pop();
360     y = stack.pop();
361     stack.push(Long.valueOf((x <= y) ? 1 : 0));
362     break;
363 case DLS_CDL_GT:
364     x = stack.pop();
365     y = stack.pop();
366     stack.push(Long.valueOf((x > y) ? 1 : 0));
367     break;
368 case DLS_CDL_GE:
369     x = stack.pop();
370     y = stack.pop();

```

```

371         stack.push(Long.valueOf((x >= y) ? 1 : 0));
372         break;
373     case DLS_CDL_EQ:
374         x = stack.pop();
375         y = stack.pop();
376         stack.push(Long.valueOf((x == y) ? 1 : 0));
377         break;
378     case DLS_CDL_NOT:
379         x = stack.pop();
380         y = stack.pop();
381         stack.push(Long.valueOf((x == 0) ? 1 : 0));
382         break;
383     case DLS_CDL_CONST:
384         stack.push(Long.valueOf(riff.readUnsignedInt()));
385         break;
386     case DLS_CDL_QUERY:
387         uuid = DLSID.read(riff);
388         stack.push(cdlQuery(uuid));
389         break;
390     case DLS_CDL_QUERY_SUPPORTED:
391         uuid = DLSID.read(riff);
392         stack.push(Long.valueOf(cdlIsQuerySupported(uuid) ? 1 : 0));
393         break;
394     default:
395         break;
396 }
397 }
398 if (stack.isEmpty())
399     return false;
400
401 return stack.pop() == 1;
402 }
403
404 private void readInfoChunk(RIFFReader riff) throws IOException {
405     info.name = null;
406     while (riff.hasNextChunk()) {
407         RIFFReader chunk = riff.nextChunk();
408         String format = chunk.getFormat();
409         if (format.equals("INAM"))
410             info.name = chunk.readString(chunk.available());
411         else if (format.equals("ICRD"))
412             info.creationDate = chunk.readString(chunk.available());
413         else if (format.equals("IENG"))
414             info.engineers = chunk.readString(chunk.available());
415         else if (format.equals("IPRD"))
416             info.product = chunk.readString(chunk.available());
417         else if (format.equals("ICOP"))
418             info.copyright = chunk.readString(chunk.available());
419         else if (format.equals("ICMT"))
420             info.comments = chunk.readString(chunk.available());
421         else if (format.equals("ISFT"))
422             info.tools = chunk.readString(chunk.available());
423         else if (format.equals("IARL"))
424             info.archival_location = chunk.readString(chunk.available());
425         else if (format.equals("IART"))
426             info.artist = chunk.readString(chunk.available());
427         else if (format.equals("ICMS"))
428             info.commissioned = chunk.readString(chunk.available());
429         else if (format.equals("IGNR"))
430             info.genre = chunk.readString(chunk.available());
431         else if (format.equals("IKEY"))
432             info.keywords = chunk.readString(chunk.available());

```



```

433     else if (format.equals("IMED"))
434         info.medium = chunk.readString(chunk.available());
435     else if (format.equals("ISBJ"))
436         info.subject = chunk.readString(chunk.available());
437     else if (format.equals("ISRC"))
438         info.source = chunk.readString(chunk.available());
439     else if (format.equals("ISRF"))
440         info.source_form = chunk.readString(chunk.available());
441     else if (format.equals("ITCH"))
442         info.technician = chunk.readString(chunk.available());
443     }
444 }
445
446 private void readLinsChunk(RIFFReader riff) throws IOException {
447     while (riff.hasNextChunk()) {
448         RIFFReader chunk = riff.nextChunk();
449         if (chunk.getFormat().equals("LIST")) {
450             if (chunk.getType().equals("ins_"))
451                 readInsChunk(chunk);
452         }
453     }
454 }
455
456 private void readInsChunk(RIFFReader riff) throws IOException {
457     DLSInstrument instrument = new DLSInstrument(this);
458
459     while (riff.hasNextChunk()) {
460         RIFFReader chunk = riff.nextChunk();
461         String format = chunk.getFormat();
462         if (format.equals("LIST")) {
463             if (chunk.getType().equals("INFO")) {
464                 readInsInfoChunk(instrument, chunk);
465             }
466             if (chunk.getType().equals("lrgn")) {
467                 while (chunk.hasNextChunk()) {
468                     RIFFReader subchunk = chunk.nextChunk();
469                     if (subchunk.getFormat().equals("LIST")) {
470                         if (subchunk.getType().equals("rgn_")) {
471                             DLSRegion split = new DLSRegion();
472                             if (readRgnChunk(split, subchunk))
473                                 instrument.getRegions().add(split);
474                         }
475                         if (subchunk.getType().equals("rgn2")) {
476                             // support for DLS level 2 regions
477                             DLSRegion split = new DLSRegion();
478                             if (readRgnChunk(split, subchunk))
479                                 instrument.getRegions().add(split);
480                         }
481                     }
482                 }
483             }
484             if (chunk.getType().equals("lart")) {
485                 List<DLSModulator> modlist = new ArrayList<DLSModulator>();
486                 while (chunk.hasNextChunk()) {
487                     RIFFReader subchunk = chunk.nextChunk();
488                     if (subchunk.getFormat().equals("cdl_")) {
489                         if (!readCdlChunk(subchunk)) {
490                             modlist.clear();
491                             break;
492                         }
493                     }
494                     if (subchunk.getFormat().equals("art1"))

```

```

495         readArt1Chunk(modlist, subchunk);
496     }
497     instrument.getModulators().addAll(modlist);
498 }
499 if (chunk.getType().equals("lar2")) {
500     // support for DLS level 2 ART
501     List<DLSModulator> modlist = new ArrayList<DLSModulator>();
502     while (chunk.hasNextChunk()) {
503         RIFFReader subchunk = chunk.nextChunk();
504         if (chunk.getFormat().equals("cdl_")) {
505             if (!readCdlChunk(chunk)) {
506                 modlist.clear();
507                 break;
508             }
509         }
510         if (subchunk.getFormat().equals("art2"))
511             readArt2Chunk(modlist, subchunk);
512     }
513     instrument.getModulators().addAll(modlist);
514 }
515 } else {
516     if (format.equals("dlid")) {
517         instrument.guid = new byte[16];
518         chunk.readFully(instrument.guid);
519     }
520     if (format.equals("insh")) {
521         chunk.readUnsignedInt(); // Read Region Count - ignored
522
523         int bank = chunk.read(); // LSB
524         bank += (chunk.read() & 127) << 7; // MSB
525         chunk.read(); // Read Reserved byte
526         int drumins = chunk.read(); // Drum Instrument
527
528         int id = chunk.read() & 127; // Read only first 7 bits
529         chunk.read(); // Read Reserved byte
530         chunk.read(); // Read Reserved byte
531         chunk.read(); // Read Reserved byte
532
533         instrument.bank = bank;
534         instrument.preset = (int) id;
535         instrument.druminstrument = (drumins & 128) > 0;
536         //System.out.println("bank="+bank+" drumkit="+drumkit
537         //                    +" id="+id);
538     }
539 }
540 }
541 }
542 instruments.add(instrument);
543 }
544
545 private void readArt1Chunk(List<DLSModulator> modulators, RIFFReader riff)
546     throws IOException {
547     long size = riff.readUnsignedInt();
548     long count = riff.readUnsignedInt();
549
550     if (size - 8 != 0)
551         riff.skipBytes(size - 8);
552
553     for (int i = 0; i < count; i++) {
554         DLSModulator modulator = new DLSModulator();
555         modulator.version = 1;
556         modulator.source = riff.readUnsignedShort();

```

```

557     modulator.control = riff.readUnsignedShort();
558     modulator.destination = riff.readUnsignedShort();
559     modulator.transform = riff.readUnsignedShort();
560     modulator.scale = riff.readInt();
561     modulators.add(modulator);
562 }
563 }
564
565 private void readArt2Chunk(List<DLSModulator> modulators, RIFFReader riff)
566     throws IOException {
567     long size = riff.readUnsignedInt();
568     long count = riff.readUnsignedInt();
569
570     if (size - 8 != 0)
571         riff.skipBytes(size - 8);
572
573     for (int i = 0; i < count; i++) {
574         DLSModulator modulator = new DLSModulator();
575         modulator.version = 2;
576         modulator.source = riff.readUnsignedShort();
577         modulator.control = riff.readUnsignedShort();
578         modulator.destination = riff.readUnsignedShort();
579         modulator.transform = riff.readUnsignedShort();
580         modulator.scale = riff.readInt();
581         modulators.add(modulator);
582     }
583 }
584
585 private Map<DLSRegion, Long> temp_rgnassign = new HashMap<DLSRegion, Long>();
586
587 private boolean readRgnChunk(DLSRegion split, RIFFReader riff)
588     throws IOException {
589     while (riff.hasNextChunk()) {
590         RIFFReader chunk = riff.nextChunk();
591         String format = chunk.getFormat();
592         if (format.equals("LIST")) {
593             if (chunk.getType().equals("lart")) {
594                 List<DLSModulator> modlist = new ArrayList<DLSModulator>();
595                 while (chunk.hasNextChunk()) {
596                     RIFFReader subchunk = chunk.nextChunk();
597                     if (subchunk.getFormat().equals("cdl_")) {
598                         if (!readCdlChunk(subchunk)) {
599                             modlist.clear();
600                             break;
601                         }
602                     }
603                     if (subchunk.getFormat().equals("art1"))
604                         readArt1Chunk(modlist, subchunk);
605                 }
606                 split.getModulators().addAll(modlist);
607             }
608             if (chunk.getType().equals("lar2")) {
609                 // support for DLS level 2 ART
610                 List<DLSModulator> modlist = new ArrayList<DLSModulator>();
611                 while (chunk.hasNextChunk()) {
612                     RIFFReader subchunk = chunk.nextChunk();
613                     if (subchunk.getFormat().equals("cdl_")) {
614                         if (!readCdlChunk(subchunk)) {
615                             modlist.clear();
616                             break;
617                         }
618                     }

```

```

619         if (subchunk.getFormat().equals("art2"))
620             readArt2Chunk(modlist, subchunk);
621     }
622     split.getModulators().addAll(modlist);
623 }
624 } else {
625
626     if (format.equals("cdl_")) {
627         if (!readCdlChunk(chunk))
628             return false;
629     }
630     if (format.equals("rgnh")) {
631         split.keyfrom = chunk.readUnsignedShort();
632         split.keyto = chunk.readUnsignedShort();
633         split.velfrom = chunk.readUnsignedShort();
634         split.velto = chunk.readUnsignedShort();
635         split.options = chunk.readUnsignedShort();
636         split.exclusiveClass = chunk.readUnsignedShort();
637     }
638     if (format.equals("wlnk")) {
639         split.fusoptions = chunk.readUnsignedShort();
640         split.phasegroup = chunk.readUnsignedShort();
641         split.channel = chunk.readUnsignedInt();
642         long sampleid = chunk.readUnsignedInt();
643         temp_rgnassign.put(split, sampleid);
644     }
645     if (format.equals("wsmp")) {
646         split.sampleoptions = new DLSSampleOptions();
647         readWsmpChunk(split.sampleoptions, chunk);
648     }
649 }
650 }
651 return true;
652 }
653
654 private void readWsmpChunk(DLSSampleOptions sampleOptions, RIFFReader riff)
655     throws IOException {
656     long size = riff.readUnsignedInt();
657     sampleOptions.unitynote = riff.readUnsignedShort();
658     sampleOptions.finetune = riff.readShort();
659     sampleOptions.attenuation = riff.readInt();
660     sampleOptions.options = riff.readUnsignedInt();
661     long loops = riff.readInt();
662
663     if (size > 20)
664         riff.skipBytes(size - 20);
665
666     for (int i = 0; i < loops; i++) {
667         DLSSampleLoop loop = new DLSSampleLoop();
668         long size2 = riff.readUnsignedInt();
669         loop.type = riff.readUnsignedInt();
670         loop.start = riff.readUnsignedInt();
671         loop.length = riff.readUnsignedInt();
672         sampleOptions.loops.add(loop);
673         if (size2 > 16)
674             riff.skipBytes(size2 - 16);
675     }
676 }
677
678 private void readInsInfoChunk(DLSInstrument dlsinstrument, RIFFReader riff)
679     throws IOException {
680     dlsinstrument.info.name = null;

```

```

681 while (riff.hasNextChunk()) {
682     RIFFReader chunk = riff.nextChunk();
683     String format = chunk.getFormat();
684     if (format.equals("INAM")) {
685         dlsinstrument.info.name = chunk.readString(chunk.available());
686     } else if (format.equals("ICRD")) {
687         dlsinstrument.info.creationDate =
688             chunk.readString(chunk.available());
689     } else if (format.equals("IENG")) {
690         dlsinstrument.info.engineers =
691             chunk.readString(chunk.available());
692     } else if (format.equals("IPRD")) {
693         dlsinstrument.info.product = chunk.readString(chunk.available());
694     } else if (format.equals("ICOP")) {
695         dlsinstrument.info.copyright =
696             chunk.readString(chunk.available());
697     } else if (format.equals("ICMT")) {
698         dlsinstrument.info.comments =
699             chunk.readString(chunk.available());
700     } else if (format.equals("ISFT")) {
701         dlsinstrument.info.tools = chunk.readString(chunk.available());
702     } else if (format.equals("IARL")) {
703         dlsinstrument.info.archival_location =
704             chunk.readString(chunk.available());
705     } else if (format.equals("IART")) {
706         dlsinstrument.info.artist = chunk.readString(chunk.available());
707     } else if (format.equals("ICMS")) {
708         dlsinstrument.info.commissioned =
709             chunk.readString(chunk.available());
710     } else if (format.equals("IGNR")) {
711         dlsinstrument.info.genre = chunk.readString(chunk.available());
712     } else if (format.equals("IKEY")) {
713         dlsinstrument.info.keywords =
714             chunk.readString(chunk.available());
715     } else if (format.equals("IMED")) {
716         dlsinstrument.info.medium = chunk.readString(chunk.available());
717     } else if (format.equals("ISBJ")) {
718         dlsinstrument.info.subject = chunk.readString(chunk.available());
719     } else if (format.equals("ISRC")) {
720         dlsinstrument.info.source = chunk.readString(chunk.available());
721     } else if (format.equals("ISRF")) {
722         dlsinstrument.info.source_form =
723             chunk.readString(chunk.available());
724     } else if (format.equals("ITCH")) {
725         dlsinstrument.info.technician =
726             chunk.readString(chunk.available());
727     }
728 }
729 }
730
731 private void readWvplChunk(RIFFReader riff) throws IOException {
732     while (riff.hasNextChunk()) {
733         RIFFReader chunk = riff.nextChunk();
734         if (chunk.getFormat().equals("LIST")) {
735             if (chunk.getType().equals("wave"))
736                 readWaveChunk(chunk);
737         }
738     }
739 }
740
741 private void readWaveChunk(RIFFReader riff) throws IOException {
742     DLSSample sample = new DLSSample(this);

```

```

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

```

```

805         read += 65536;
806     } else {
807         chunk.readFully(buffer, read, avail - read);
808         read = avail;
809     }
810 }
811 }
812 }
813
814 if (format.equals("wsmp")) {
815     sample.sampleoptions = new DLSSampleOptions();
816     readWsmpChunk(sample.sampleoptions, chunk);
817 }
818 }
819 }
820
821 samples.add(sample);
822
823 }
824
825 private void readWaveInfoChunk(DLSSample dlssample, RIFFReader riff)
826     throws IOException {
827     dlssample.info.name = null;
828     while (riff.hasNextChunk()) {
829         RIFFReader chunk = riff.nextChunk();
830         String format = chunk.getFormat();
831         if (format.equals("INAM")) {
832             dlssample.info.name = chunk.readString(chunk.available());
833         } else if (format.equals("ICRD")) {
834             dlssample.info.creationDate =
835                 chunk.readString(chunk.available());
836         } else if (format.equals("IENG")) {
837             dlssample.info.engineers = chunk.readString(chunk.available());
838         } else if (format.equals("IPRD")) {
839             dlssample.info.product = chunk.readString(chunk.available());
840         } else if (format.equals("ICOP")) {
841             dlssample.info.copyright = chunk.readString(chunk.available());
842         } else if (format.equals("ICMT")) {
843             dlssample.info.comments = chunk.readString(chunk.available());
844         } else if (format.equals("ISFT")) {
845             dlssample.info.tools = chunk.readString(chunk.available());
846         } else if (format.equals("IARL")) {
847             dlssample.info.archival_location =
848                 chunk.readString(chunk.available());
849         } else if (format.equals("IART")) {
850             dlssample.info.artist = chunk.readString(chunk.available());
851         } else if (format.equals("ICMS")) {
852             dlssample.info.commissioned =
853                 chunk.readString(chunk.available());
854         } else if (format.equals("IGNR")) {
855             dlssample.info.genre = chunk.readString(chunk.available());
856         } else if (format.equals("IKEY")) {
857             dlssample.info.keywords = chunk.readString(chunk.available());
858         } else if (format.equals("IMED")) {
859             dlssample.info.medium = chunk.readString(chunk.available());
860         } else if (format.equals("ISBJ")) {
861             dlssample.info.subject = chunk.readString(chunk.available());
862         } else if (format.equals("ISRC")) {
863             dlssample.info.source = chunk.readString(chunk.available());
864         } else if (format.equals("ISRF")) {
865             dlssample.info.source_form = chunk.readString(chunk.available());
866         } else if (format.equals("ITCH")) {

```

```

867         dlssample.info.technician = chunk.readString(chunk.available());
868     }
869 }
870
871
872 public void save(String name) throws IOException {
873     writeSoundbank(new RIFFWriter(name, "DLS_"));
874 }
875
876 public void save(File file) throws IOException {
877     writeSoundbank(new RIFFWriter(file, "DLS_"));
878 }
879
880 public void save(OutputStream out) throws IOException {
881     writeSoundbank(new RIFFWriter(out, "DLS_"));
882 }
883
884 private void writeSoundbank(RIFFWriter writer) throws IOException {
885     RIFFWriter colh_chunk = writer.writeChunk("colh");
886     colh_chunk.writeUnsignedInt(instruments.size());
887
888     if (major != -1 && minor != -1) {
889         RIFFWriter vers_chunk = writer.writeChunk("vers");
890         vers_chunk.writeUnsignedInt(major);
891         vers_chunk.writeUnsignedInt(minor);
892     }
893
894     writeInstruments(writer.writeList("lins"));
895
896     RIFFWriter ptbl = writer.writeChunk("ptbl");
897     ptbl.writeUnsignedInt(8);
898     ptbl.writeUnsignedInt(samples.size());
899     long ptbl_offset = writer.getFilePointer();
900     for (int i = 0; i < samples.size(); i++)
901         ptbl.writeUnsignedInt(0);
902
903     RIFFWriter wvpl = writer.writeList("wvpl");
904     long off = wvpl.getFilePointer();
905     List<Long> offsettable = new ArrayList<Long>();
906     for (DLSSample sample : samples) {
907         offsettable.add(Long.valueOf(wvpl.getFilePointer() - off));
908         writeSample(wvpl.writeList("wave"), sample);
909     }
910
911     // small cheat, we are going to rewrite data back in wvpl
912     long bak = writer.getFilePointer();
913     writer.seek(ptbl_offset);
914     writer.setWriteOverride(true);
915     for (Long offset : offsettable)
916         writer.writeUnsignedInt(offset.longValue());
917     writer.setWriteOverride(false);
918     writer.seek(bak);
919
920     writeInfo(writer.writeList("INFO"), info);
921
922     writer.close();
923 }
924
925 private void writeSample(RIFFWriter writer, DLSSample sample)
926     throws IOException {
927
928     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);

```

```

991     }
992 } else {
993     RIFFWriter data_chunk = writer.writeChunk("data");
994     ModelByteBuffer databuff = sample.getDataBuffer();
995     databuff.writeTo(data_chunk);
996     /*
997     data_chunk.write(databuff.array(),
998     databuff.arrayOffset(),
999     databuff.capacity());
1000     */
1001 }
1002
1003 writeInfo(writer.writeList("INFO"), sample.info);
1004 }
1005
1006 private void writeInstruments(RIFFWriter writer) throws IOException {
1007     for (DLSInstrument instrument : instruments) {
1008         writeInstrument(writer.writeList("ins_"), instrument);
1009     }
1010 }
1011
1012 private void writeInstrument(RIFFWriter writer, DLSInstrument instrument)
1013     throws IOException {
1014
1015     int art1_count = 0;
1016     int art2_count = 0;
1017     for (DLSModulator modulator : instrument.getModulators()) {
1018         if (modulator.version == 1)
1019             art1_count++;
1020         if (modulator.version == 2)
1021             art2_count++;
1022     }
1023     for (DLSRegion region : instrument.regions) {
1024         for (DLSModulator modulator : region.getModulators()) {
1025             if (modulator.version == 1)
1026                 art1_count++;
1027             if (modulator.version == 2)
1028                 art2_count++;
1029         }
1030     }
1031
1032     int version = 1;
1033     if (art2_count > 0)
1034         version = 2;
1035
1036     RIFFWriter insh_chunk = writer.writeChunk("insh");
1037     insh_chunk.writeUnsignedInt(instrument.getRegions().size());
1038     insh_chunk.writeUnsignedInt(instrument.bank +
1039         (instrument.druminstrument ? 2147483648L : 0));
1040     insh_chunk.writeUnsignedInt(instrument.preset);
1041
1042     RIFFWriter lrgn = writer.writeList("lrgn");
1043     for (DLSRegion region : instrument.regions)
1044         writeRegion(lrgn, region, version);
1045
1046     writeArticulators(writer, instrument.getModulators());
1047
1048     writeInfo(writer.writeList("INFO"), instrument.info);
1049 }
1050
1051 private void writeArticulators(RIFFWriter writer,

```

```

1053         List<DLSModulator> modulators) throws IOException {
1054     int art1_count = 0;
1055     int art2_count = 0;
1056     for (DLSModulator modulator : modulators) {
1057         if (modulator.version == 1)
1058             art1_count++;
1059         if (modulator.version == 2)
1060             art2_count++;
1061     }
1062     if (art1_count > 0) {
1063         RIFFWriter lar1 = writer.writeList("lar1");
1064         RIFFWriter art1 = lar1.writeChunk("art1");
1065         art1.writeUnsignedInt(8);
1066         art1.writeUnsignedInt(art1_count);
1067         for (DLSModulator modulator : modulators) {
1068             if (modulator.version == 1) {
1069                 art1.writeUnsignedShort(modulator.source);
1070                 art1.writeUnsignedShort(modulator.control);
1071                 art1.writeUnsignedShort(modulator.destination);
1072                 art1.writeUnsignedShort(modulator.transform);
1073                 art1.writeInt(modulator.scale);
1074             }
1075         }
1076     }
1077     if (art2_count > 0) {
1078         RIFFWriter lar2 = writer.writeList("lar2");
1079         RIFFWriter art2 = lar2.writeChunk("art2");
1080         art2.writeUnsignedInt(8);
1081         art2.writeUnsignedInt(art2_count);
1082         for (DLSModulator modulator : modulators) {
1083             if (modulator.version == 2) {
1084                 art2.writeUnsignedShort(modulator.source);
1085                 art2.writeUnsignedShort(modulator.control);
1086                 art2.writeUnsignedShort(modulator.destination);
1087                 art2.writeUnsignedShort(modulator.transform);
1088                 art2.writeInt(modulator.scale);
1089             }
1090         }
1091     }
1092 }
1093
1094 private void writeRegion(RIFFWriter writer, DLSRegion region, int version)
1095     throws IOException {
1096     RIFFWriter rgns = null;
1097     if (version == 1)
1098         rgns = writer.writeList("rgn1");
1099     if (version == 2)
1100         rgns = writer.writeList("rgn2");
1101     if (rgns == null)
1102         return;
1103
1104     RIFFWriter rgnh = rgns.writeChunk("rgnh");
1105     rgnh.writeUnsignedShort(region.keyfrom);
1106     rgnh.writeUnsignedShort(region.keyto);
1107     rgnh.writeUnsignedShort(region.velfrom);
1108     rgnh.writeUnsignedShort(region.velto);
1109     rgnh.writeUnsignedShort(region.options);
1110     rgnh.writeUnsignedShort(region.exclusiveClass);
1111
1112     if (region.sampleoptions != null)
1113         writeSampleOptions(rgns.writeChunk("wsmp"), region.sampleoptions);
1114

```

```

1115     if (region.sample != null) {
1116         if (samples.indexOf(region.sample) != -1) {
1117             RIFFWriter wlnk = rgns.writeChunk("wlnk");
1118             wlnk.writeUnsignedShort(region.fusoptions);
1119             wlnk.writeUnsignedShort(region.phasegroup);
1120             wlnk.writeUnsignedInt(region.channel);
1121             wlnk.writeUnsignedInt(samples.indexOf(region.sample));
1122         }
1123     }
1124     writeArticulators(rgns, region.getModulators());
1125     rgns.close();
1126 }
1127
1128 private void writeSampleOptions(RIFFWriter wsmg,
1129     DLSSampleOptions sampleoptions) throws IOException {
1130     wsmg.writeUnsignedInt(20);
1131     wsmg.writeUnsignedShort(sampleoptions.unitynote);
1132     wsmg.writeShort(sampleoptions.finetune);
1133     wsmg.writeInt(sampleoptions.attenuation);
1134     wsmg.writeUnsignedInt(sampleoptions.options);
1135     wsmg.writeInt(sampleoptions.loops.size());
1136
1137     for (DLSSampleLoop loop : sampleoptions.loops) {
1138         wsmg.writeUnsignedInt(16);
1139         wsmg.writeUnsignedInt(loop.type);
1140         wsmg.writeUnsignedInt(loop.start);
1141         wsmg.writeUnsignedInt(loop.length);
1142     }
1143 }
1144
1145 private void writeInfoStringChunk(RIFFWriter writer,
1146     String name, String value) throws IOException {
1147     if (value == null)
1148         return;
1149     RIFFWriter chunk = writer.writeChunk(name);
1150     chunk.writeString(value);
1151     int len = value.getBytes("ascii").length;
1152     chunk.write(0);
1153     len++;
1154     if (len % 2 != 0)
1155         chunk.write(0);
1156 }
1157
1158 private void writeInfo(RIFFWriter writer, DLSInfo info) throws IOException {
1159     writeInfoStringChunk(writer, "INAM", info.name);
1160     writeInfoStringChunk(writer, "ICRD", info.creationDate);
1161     writeInfoStringChunk(writer, "IENG", info.engineers);
1162     writeInfoStringChunk(writer, "IPRD", info.product);
1163     writeInfoStringChunk(writer, "ICOP", info.copyright);
1164     writeInfoStringChunk(writer, "ICMT", info.comments);
1165     writeInfoStringChunk(writer, "ISFT", info.tools);
1166     writeInfoStringChunk(writer, "IARL", info.archival_location);
1167     writeInfoStringChunk(writer, "IART", info.artist);
1168     writeInfoStringChunk(writer, "ICMS", info.commissioned);
1169     writeInfoStringChunk(writer, "IGNR", info.genre);
1170     writeInfoStringChunk(writer, "IKEY", info.keywords);
1171     writeInfoStringChunk(writer, "IMED", info.medium);
1172     writeInfoStringChunk(writer, "ISBJ", info.subject);
1173     writeInfoStringChunk(writer, "ISRC", info.source);
1174     writeInfoStringChunk(writer, "ISRF", info.source_form);
1175     writeInfoStringChunk(writer, "ITCH", info.technician);
1176 }

```

```

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();

```

```

1239         if (program == program2 && bank == bank2) {
1240             boolean percussion2 = false;
1241             if (patch2 instanceof ModelPatch)
1242                 percussion2 = ((ModelPatch) patch2).isPercussion();
1243             if (percussion == percussion2)
1244                 return instrument;
1245         }
1246     }
1247     return null;
1248 }
1249
1250 public void addResource(SoundbankResource resource) {
1251     if (resource instanceof DLSInstrument)
1252         instruments.add((DLSInstrument) resource);
1253     if (resource instanceof DLSSample)
1254         samples.add((DLSSample) resource);
1255 }
1256
1257 public void removeResource(SoundbankResource resource) {
1258     if (resource instanceof DLSInstrument)
1259         instruments.remove((DLSInstrument) resource);
1260     if (resource instanceof DLSSample)
1261         samples.remove((DLSSample) resource);
1262 }
1263
1264 public void addInstrument(DLSInstrument resource) {
1265     instruments.add(resource);
1266 }
1267
1268 public void removeInstrument(DLSInstrument resource) {
1269     instruments.remove(resource);
1270 }
1271
1272 public long getMajor() {
1273     return major;
1274 }
1275
1276 public void setMajor(long major) {
1277     this.major = major;
1278 }
1279
1280 public long getMinor() {
1281     return minor;
1282 }
1283
1284 public void setMinor(long minor) {
1285     this.minor = minor;
1286 }
1287 }

```

30 com/sun/media/sound/DLSSoundbankReader.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
26 package com.sun.media.sound;
27
28 import java.io.File;
29 import java.io.IOException;
30 import java.io.InputStream;
31 import java.net.URL;
32 import javax.sound.midi.InvalidMidiDataException;
33 import javax.sound.midi.Soundbank;
34 import javax.sound.midi.spi.SoundbankReader;
35
36 /**
37 * This class is used to connect the DLSSoundBank class
38 * to the SoundbankReader SPI interface.
39 *
40 * @author Karl Helgason
41 */
42 public class DLSSoundbankReader extends SoundbankReader {
43
44     public Soundbank getSoundbank(URL url)
45         throws InvalidMidiDataException, IOException {
46         try {
47             return new DLSSoundbank(url);
48         } catch (RIFFInvalidFormatException e) {
49             return null;
50         } catch (IOException ioe) {
51             return null;
52         }
53     }
54
55     public Soundbank getSoundbank(InputStream stream)
56         throws InvalidMidiDataException, IOException {
57         try {
58             stream.mark(512);
59             return new DLSSoundbank(stream);
60         } 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

```
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.Random;
28
29 import javax.sound.midi.Patch;
30 import javax.sound.sampled.AudioFormat;
31
32 /**
33  * Emergency Soundbank generator.
34  * Used when no other default soundbank can be found.
35  *
36  * @author Karl Helgason
37  */
38 public class EmergencySoundbank {
39
40     private final static String[] general_midi_instruments = {
41         "Acoustic_Grand_Piano",
42         "Bright_Acoustic_Piano",
43         "Electric_Grand_Piano",
44         "Honky-tonk_Piano",
45         "Electric_Piano_1",
46         "Electric_Piano_2",
47         "Harpsichord",
48         "Clavi",
49         "Celesta",
50         "Glockenspiel",
51         "Music_Box",
52         "Vibraphone",
53         "Marimba",
54         "Xylophone",
55         "Tubular_Bells",
56         "Dulcimer",
57         "Drawbar_Organ",
58         "Percussive_Organ",
59         "Rock_Organ",
60         "Church_Organ",
```

```

61 "Reed_Organ",
62 "Accordion",
63 "Harmonica",
64 "Tango_Accordion",
65 "Acoustic_Guitar_(nylon)",
66 "Acoustic_Guitar_(steel)",
67 "Electric_Guitar_(jazz)",
68 "Electric_Guitar_(clean)",
69 "Electric_Guitar_(muted)",
70 "Overdriven_Guitar",
71 "Distortion_Guitar",
72 "Guitar_harmonics",
73 "Acoustic_Bass",
74 "Electric_Bass_(finger)",
75 "Electric_Bass_(pick)",
76 "Fretless_Bass",
77 "Slap_Bass_1",
78 "Slap_Bass_2",
79 "Synth_Bass_1",
80 "Synth_Bass_2",
81 "Violin",
82 "Viola",
83 "Cello",
84 "Contrabass",
85 "Tremolo_Strings",
86 "Pizzicato_Strings",
87 "Orchestral_Harp",
88 "Timpani",
89 "String_Ensemble_1",
90 "String_Ensemble_2",
91 "SynthStrings_1",
92 "SynthStrings_2",
93 "Choir_Aahs",
94 "Voice_Oohs",
95 "Synth_Voice",
96 "Orchestra_Hit",
97 "Trumpet",
98 "Trombone",
99 "Tuba",
100 "Muted_Trumpet",
101 "French_Horn",
102 "Brass_Section",
103 "SynthBrass_1",
104 "SynthBrass_2",
105 "Soprano_Sax",
106 "Alto_Sax",
107 "Tenor_Sax",
108 "Baritone_Sax",
109 "Oboe",
110 "English_Horn",
111 "Bassoon",
112 "Clarinet",
113 "Piccolo",
114 "Flute",
115 "Recorder",
116 "Pan_Flute",
117 "Blown_Bottle",
118 "Shakuhachi",
119 "Whistle",
120 "Ocarina",
121 "Lead_1_(square)",
122 "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);
}

```

```

185 SF2Layer closed_hihat = new_closed_hihat(sf2);
186 SF2Layer crash_cymbal = new_crash_cymbal(sf2);
187 SF2Layer side_stick = new_side_stick(sf2);
188
189 SF2Layer[] drums = new SF2Layer[128];
190 drums[35] = bass_drum;
191 drums[36] = bass_drum;
192 drums[38] = snare_drum;
193 drums[40] = snare_drum;
194 drums[41] = tom;
195 drums[43] = tom;
196 drums[45] = tom;
197 drums[47] = tom;
198 drums[48] = tom;
199 drums[50] = tom;
200 drums[42] = closed_hihat;
201 drums[44] = closed_hihat;
202 drums[46] = open_hihat;
203 drums[49] = crash_cymbal;
204 drums[51] = crash_cymbal;
205 drums[52] = crash_cymbal;
206 drums[55] = crash_cymbal;
207 drums[57] = crash_cymbal;
208 drums[59] = crash_cymbal;
209
210 // Use side_stick for missing drums:
211 drums[37] = side_stick;
212 drums[39] = side_stick;
213 drums[53] = side_stick;
214 drums[54] = side_stick;
215 drums[56] = side_stick;
216 drums[58] = side_stick;
217 drums[69] = side_stick;
218 drums[70] = side_stick;
219 drums[75] = side_stick;
220 drums[60] = side_stick;
221 drums[61] = side_stick;
222 drums[62] = side_stick;
223 drums[63] = side_stick;
224 drums[64] = side_stick;
225 drums[65] = side_stick;
226 drums[66] = side_stick;
227 drums[67] = side_stick;
228 drums[68] = side_stick;
229 drums[71] = side_stick;
230 drums[72] = side_stick;
231 drums[73] = side_stick;
232 drums[74] = side_stick;
233 drums[76] = side_stick;
234 drums[77] = side_stick;
235 drums[78] = side_stick;
236 drums[79] = side_stick;
237 drums[80] = side_stick;
238 drums[81] = side_stick;
239
240
241 SF2Instrument drum_instrument = new SF2Instrument(sf2);
242 drum_instrument.setName("Standard_Kit");
243 drum_instrument.setPatch(new ModelPatch(0, 0, true));
244 sf2.addInstrument(drum_instrument);
245 for (int i = 0; i < drums.length; i++) {
246     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  * melodic instruments
257  */
258
259 SF2Layer gpiano = new_gpiano(sf2);
260 SF2Layer gpiano2 = new_gpiano2(sf2);
261 SF2Layer gpiano_hammer = new_piano_hammer(sf2);
262 SF2Layer piano1 = new_piano1(sf2);
263 SF2Layer epiano1 = new_epiano1(sf2);
264 SF2Layer epiano2 = new_epiano2(sf2);
265
266 SF2Layer guitar = new_guitar1(sf2);
267 SF2Layer guitar_pick = new_guitar_pick(sf2);
268 SF2Layer guitar_dist = new_guitar_dist(sf2);
269 SF2Layer bass1 = new_bass1(sf2);
270 SF2Layer bass2 = new_bass2(sf2);
271 SF2Layer synthbass = new_synthbass(sf2);
272 SF2Layer string2 = new_string2(sf2);
273 SF2Layer orchhit = new_orchhit(sf2);
274 SF2Layer choir = new_choir(sf2);
275 SF2Layer solostring = new_solostring(sf2);
276 SF2Layer organ = new_organ(sf2);
277 SF2Layer ch_organ = new_ch_organ(sf2);
278 SF2Layer bell = new_bell(sf2);
279 SF2Layer flute = new_flute(sf2);
280
281 SF2Layer timpani = new_timpani(sf2);
282 SF2Layer melodic_toms = new_melodic_toms(sf2);
283 SF2Layer trumpet = new_trumpet(sf2);
284 SF2Layer trombone = new_trombone(sf2);
285 SF2Layer brass_section = new_brass_section(sf2);
286 SF2Layer horn = new_horn(sf2);
287 SF2Layer sax = new_sax(sf2);
288 SF2Layer oboe = new_oboe(sf2);
289 SF2Layer bassoon = new_bassoon(sf2);
290 SF2Layer clarinet = new_clarinet(sf2);
291 SF2Layer reverse_cymbal = new_reverse_cymbal(sf2);
292
293 SF2Layer defaultsound = piano1;
294
295 newInstrument(sf2, "Piano", new Patch(0, 0), gpiano, gpiano_hammer);
296 newInstrument(sf2, "Piano", new Patch(0, 1), gpiano2, gpiano_hammer);
297 newInstrument(sf2, "Piano", new Patch(0, 2), piano1);
298 {
299     SF2Instrument ins = newInstrument(sf2, "Honky-tonk_Piano",
300         new Patch(0, 3), piano1, piano1);
301     SF2InstrumentRegion region = ins.getRegions().get(0);
302     region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 80);
303     region.putInteger(SF2Region.GENERATOR_FINETUNE, 30);
304     region = ins.getRegions().get(1);
305     region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 30);
306 }
307 newInstrument(sf2, "Rhodes", new Patch(0, 4), epiano2);
308

```

```

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, "Fretless_Bass", new Patch(0, 35), bass2);
340 newInstrument(sf2, "Fretless_Bass", new Patch(0, 36), bass2);
341 newInstrument(sf2, "Fretless_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);

```

```

371 }
372 newInstrument(sf2, "Trumpet", new Patch(0, 56), trumpet);
373 newInstrument(sf2, "Trombone", new Patch(0, 57), trombone);
374 newInstrument(sf2, "Trombone", new Patch(0, 58), trombone);
375 newInstrument(sf2, "Trumpet", new Patch(0, 59), trumpet);
376 newInstrument(sf2, "Horn", new Patch(0, 60), horn);
377 newInstrument(sf2, "Brass_Section", new Patch(0, 61), brass_section);
378 newInstrument(sf2, "Brass_Section", new Patch(0, 62), brass_section);
379 newInstrument(sf2, "Brass_Section", new Patch(0, 63), brass_section);
380 newInstrument(sf2, "Sax", new Patch(0, 64), sax);
381 newInstrument(sf2, "Sax", new Patch(0, 65), sax);
382 newInstrument(sf2, "Sax", new Patch(0, 66), sax);
383 newInstrument(sf2, "Sax", new Patch(0, 67), sax);
384 newInstrument(sf2, "Oboe", new Patch(0, 68), oboe);
385 newInstrument(sf2, "Horn", new Patch(0, 69), horn);
386 newInstrument(sf2, "Bassoon", new Patch(0, 70), bassoon);
387 newInstrument(sf2, "Clarinet", new Patch(0, 71), clarinet);
388 newInstrument(sf2, "Flute", new Patch(0, 72), flute);
389 newInstrument(sf2, "Flute", new Patch(0, 73), flute);
390 newInstrument(sf2, "Flute", new Patch(0, 74), flute);
391 newInstrument(sf2, "Flute", new Patch(0, 75), flute);
392 newInstrument(sf2, "Flute", new Patch(0, 76), flute);
393 newInstrument(sf2, "Flute", new Patch(0, 77), flute);
394 newInstrument(sf2, "Flute", new Patch(0, 78), flute);
395 newInstrument(sf2, "Flute", new Patch(0, 79), flute);
396 newInstrument(sf2, "Organ", new Patch(0, 80), organ);
397 newInstrument(sf2, "Organ", new Patch(0, 81), organ);
398 newInstrument(sf2, "Flute", new Patch(0, 82), flute);
399 newInstrument(sf2, "Organ", new Patch(0, 83), organ);
400 newInstrument(sf2, "Organ", new Patch(0, 84), organ);
401 newInstrument(sf2, "Choir", new Patch(0, 85), choir);
402 newInstrument(sf2, "Organ", new Patch(0, 86), organ);
403 newInstrument(sf2, "Organ", new Patch(0, 87), organ);
404 newInstrument(sf2, "Synth_Strings", new Patch(0, 88), string2);
405 newInstrument(sf2, "Organ", new Patch(0, 89), organ);
406 newInstrument(sf2, "Def", new Patch(0, 90), defaultsound);
407 newInstrument(sf2, "Choir", new Patch(0, 91), choir);
408 newInstrument(sf2, "Organ", new Patch(0, 92), organ);
409 newInstrument(sf2, "Organ", new Patch(0, 93), organ);
410 newInstrument(sf2, "Organ", new Patch(0, 94), organ);
411 newInstrument(sf2, "Organ", new Patch(0, 95), organ);
412 newInstrument(sf2, "Organ", new Patch(0, 96), organ);
413 newInstrument(sf2, "Organ", new Patch(0, 97), organ);
414 newInstrument(sf2, "Bell", new Patch(0, 98), bell);
415 newInstrument(sf2, "Organ", new Patch(0, 99), organ);
416 newInstrument(sf2, "Organ", new Patch(0, 100), organ);
417 newInstrument(sf2, "Organ", new Patch(0, 101), organ);
418 newInstrument(sf2, "Def", new Patch(0, 102), defaultsound);
419 newInstrument(sf2, "Synth_Strings", new Patch(0, 103), string2);
420 newInstrument(sf2, "Def", new Patch(0, 104), defaultsound);
421 newInstrument(sf2, "Def", new Patch(0, 105), defaultsound);
422 newInstrument(sf2, "Def", new Patch(0, 106), defaultsound);
423 newInstrument(sf2, "Def", new Patch(0, 107), defaultsound);
424 newInstrument(sf2, "Marimba", new Patch(0, 108), bell);
425 newInstrument(sf2, "Sax", new Patch(0, 109), sax);
426 newInstrument(sf2, "Solo_String", new Patch(0, 110), string2, solostring);
427 newInstrument(sf2, "Oboe", new Patch(0, 111), oboe);
428 newInstrument(sf2, "Bell", new Patch(0, 112), bell);
429 newInstrument(sf2, "Melodic_Toms", new Patch(0, 113), melodic_toms);
430 newInstrument(sf2, "Marimba", new Patch(0, 114), bell);
431 newInstrument(sf2, "Melodic_Toms", new Patch(0, 115), melodic_toms);
432 newInstrument(sf2, "Melodic_Toms", new Patch(0, 116), melodic_toms);

```

```

433 newInstrument(sf2, "Melodic_Toms", new Patch(0, 117), melodic_toms);
434 newInstrument(sf2, "Reverse_Cymbal", new Patch(0, 118), reverse_cymbal);
435 newInstrument(sf2, "Reverse_Cymbal", new Patch(0, 119), reverse_cymbal);
436 newInstrument(sf2, "Guitar", new Patch(0, 120), guitar);
437 newInstrument(sf2, "Def", new Patch(0, 121), defaultsound);
438 {
439     SF2Instrument ins = newInstrument(sf2, "Seashore/Reverse_Cymbal",
440         new Patch(0, 122), reverse_cymbal);
441     region = ins.getRegions().get(0);
442     region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
443     region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 18500);
444     region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 4500);
445     region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, -4500);
446 }
447 {
448     SF2Instrument ins = newInstrument(sf2, "Bird/Flute",
449         new Patch(0, 123), flute);
450     region = ins.getRegions().get(0);
451     region.putInteger(SF2Region.GENERATOR_COARSETUNE, 24);
452     region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, -3000);
453     region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
454 }
455 newInstrument(sf2, "Def", new Patch(0, 124), side_stick);
456 {
457     SF2Instrument ins = newInstrument(sf2, "Seashore/Reverse_Cymbal",
458         new Patch(0, 125), reverse_cymbal);
459     region = ins.getRegions().get(0);
460     region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
461     region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 18500);
462     region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 4500);
463     region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, -4500);
464 }
465 newInstrument(sf2, "Applause/crash_cymbal",
466     new Patch(0, 126), crash_cymbal);
467 newInstrument(sf2, "Gunshot/side_stick", new Patch(0, 127), side_stick);
468
469 for (SF2Instrument instrument : sf2.getInstruments()) {
470     Patch patch = instrument.getPatch();
471     if (patch instanceof ModelPatch) {
472         if (((ModelPatch) patch).isPercussion())
473             continue;
474     }
475     instrument.setName(general_midi_instruments[patch.getProgram()]);
476 }
477
478 return sf2;
479
480 }
481
482 public static SF2Layer new_bell(SF2Soundbank sf2) {
483     Random random = new Random(102030201);
484     int x = 8;
485     int fftsize = 4096 * x;
486     double[] data = new double[fftsize * 2];
487     double base = x * 25;
488     double start_w = 0.01;
489     double end_w = 0.05;
490     double start_a = 0.2;
491     double end_a = 0.00001;
492     double a = start_a;
493     double a_step = Math.pow(end_a / start_a, 1.0 / 40.0);
494     for (int i = 0; i < 40; i++) {

```



```

495         double detune = 1 + (random.nextDouble() * 2 - 1) * 0.01;
496         double w = start_w + (end_w - start_w) * (i / 40.0);
497         complexGaussianDist(data, base * (i + 1) * detune, w, a);
498         a *= a_step;
499     }
500     SF2Sample sample = newSimpleFFTSample(sf2, "EPiano", data, base);
501     SF2Layer layer = newLayer(sf2, "EPiano", sample);
502     SF2Region region = layer.getRegions().get(0);
503     region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
504     region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -12000);
505     region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 0);
506     region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
507     region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
508     region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, 1200);
509     region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
510     region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, -9000);
511     region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 16000);
512     return layer;
513 }
514
515 public static SF2Layer new_guitar1(SF2Soundbank sf2) {
516
517     int x = 8;
518     int fftsize = 4096 * x;
519     double[] data = new double[fftsize * 2];
520     double base = x * 25;
521     double start_w = 0.01;
522     double end_w = 0.01;
523     double start_a = 2;
524     double end_a = 0.01;
525     double a = start_a;
526     double a_step = Math.pow(end_a / start_a, 1.0 / 40.0);
527
528     double[] aa = new double[40];
529     for (int i = 0; i < 40; i++) {
530         aa[i] = a;
531         a *= a_step;
532     }
533
534     aa[0] = 2;
535     aa[1] = 0.5;
536     aa[2] = 0.45;
537     aa[3] = 0.2;
538     aa[4] = 1;
539     aa[5] = 0.5;
540     aa[6] = 2;
541     aa[7] = 1;
542     aa[8] = 0.5;
543     aa[9] = 1;
544     aa[9] = 0.5;
545     aa[10] = 0.2;
546     aa[11] = 1;
547     aa[12] = 0.7;
548     aa[13] = 0.5;
549     aa[14] = 1;
550
551     for (int i = 0; i < 40; i++) {
552         double w = start_w + (end_w - start_w) * (i / 40.0);
553         complexGaussianDist(data, base * (i + 1), w, aa[i]);
554     }
555
556     SF2Sample sample = newSimpleFFTSample(sf2, "Guitar", data, base);

```

```

557 SF2Layer layer = newLayer(sf2, "Guitar", sample);
558 SF2Region region = layer.getRegions().get(0);
559 region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
560 region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -12000);
561 region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 0);
562 region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 2400);
563 region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
564
565 region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, -100);
566 region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
567 region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, -6000);
568 region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 16000);
569 region.putInteger(SF2Region.GENERATOR_INITIALATTENUATION, -20);
570 return layer;
571 }
572
573 public static SF2Layer new_guitar_dist(SF2Soundbank sf2) {
574
575     int x = 8;
576     int fftsize = 4096 * x;
577     double[] data = new double[fftsize * 2];
578     double base = x * 25;
579     double start_w = 0.01;
580     double end_w = 0.01;
581     double start_a = 2;
582     double end_a = 0.01;
583     double a = start_a;
584     double a_step = Math.pow(end_a / start_a, 1.0 / 40.0);
585
586     double[] aa = new double[40];
587     for (int i = 0; i < 40; i++) {
588         aa[i] = a;
589         a *= a_step;
590     }
591
592     aa[0] = 5;
593     aa[1] = 2;
594     aa[2] = 0.45;
595     aa[3] = 0.2;
596     aa[4] = 1;
597     aa[5] = 0.5;
598     aa[6] = 2;
599     aa[7] = 1;
600     aa[8] = 0.5;
601     aa[9] = 1;
602     aa[9] = 0.5;
603     aa[10] = 0.2;
604     aa[11] = 1;
605     aa[12] = 0.7;
606     aa[13] = 0.5;
607     aa[14] = 1;
608
609     for (int i = 0; i < 40; i++) {
610         double w = start_w + (end_w - start_w) * (i / 40.0);
611         complexGaussianDist(data, base * (i + 1), w, aa[i]);
612     }
613
614     SF2Sample sample = newSimpleFFTSample_dist(sf2, "Distorted_Guitar",
615         data, base, 10000.0);
616
617
618

```

```

619 SF2Layer layer = newLayer(sf2, "Distorted_Guitar", sample);
620 SF2Region region = layer.getRegions().get(0);
621 region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
622 region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -12000);
623 region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 0);
624 //region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 2400);
625 //region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 200);
626
627 //region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, -100);
628 //region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
629 //region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, -1000);
630 region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 8000);
631 //region.putInteger(SF2Region.GENERATOR_INITIALATTENUATION, -20);
632 return layer;
633 }
634
635 public static SF2Layer new_guitar_pick(SF2Soundbank sf2) {
636
637     double datab[];
638
639     // Make treble part
640     {
641         int m = 2;
642         int fftlen = 4096 * m;
643         double[] data = new double[2 * fftlen];
644         Random random = new Random(3049912);
645         for (int i = 0; i < data.length; i += 2)
646             data[i] = (2.0 * (random.nextDouble() - 0.5));
647         fft(data);
648         // Remove all negative frequency
649         for (int i = fftlen / 2; i < data.length; i++)
650             data[i] = 0;
651         for (int i = 0; i < 2048 * m; i++) {
652             data[i] *= Math.exp(-Math.abs((i - 23) / ((double) m)) * 1.2)
653                 + Math.exp(-Math.abs((i - 40) / ((double) m)) * 0.9);
654         }
655         randomPhase(data, new Random(3049912));
656         ifft(data);
657         normalize(data, 0.8);
658         data = realPart(data);
659         double gain = 1.0;
660         for (int i = 0; i < data.length; i++) {
661             data[i] *= gain;
662             gain *= 0.9994;
663         }
664         datab = data;
665
666         fadeUp(data, 80);
667     }
668
669     SF2Sample sample = newSimpleDrumSample(sf2, "Guitar_Noise", datab);
670
671     SF2Layer layer = new SF2Layer(sf2);
672     layer.setName("Guitar_Noise");
673
674     SF2GlobalRegion global = new SF2GlobalRegion();
675     layer.setGlobalZone(global);
676     sf2.addResource(layer);
677
678     SF2LayerRegion region = new SF2LayerRegion();
679     region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 12000);
680     //region.putInteger(SF2Region.GENERATOR_SCALETUNING, 0);

```

```

681 //      region.putInteger(SF2Region.GENERATOR_INITIALATTENUATION, -100);
682 /*
683         region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, 0);
684         region.putInteger(SF2Region.GENERATOR_SUSTAINMODENV, 1000);
685         region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
686         region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, -11000);
687         region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 12000);
688         */
689
690         region.setSample(sample);
691         layer.getRegions().add(region);
692
693         return layer;
694     }
695
696     public static SF2Layer new_gpiano(SF2Soundbank sf2) {
697         //Random random = new Random(302030201);
698         int x = 8;
699         int fftsize = 4096 * x;
700         double[] data = new double[fftsize * 2];
701         double base = x * 25;
702         double start_a = 0.2;
703         double end_a = 0.001;
704         double a = start_a;
705         double a_step = Math.pow(end_a / start_a, 1.0 / 15.0);
706
707         double[] aa = new double[30];
708         for (int i = 0; i < 30; i++) {
709             aa[i] = a;
710             a *= a_step;
711         }
712
713         aa[0] *= 2;
714         //aa[2] *= 0.1;
715         aa[4] *= 2;
716
717
718         aa[12] *= 0.9;
719         aa[13] *= 0.7;
720         for (int i = 14; i < 30; i++) {
721             aa[i] *= 0.5;
722         }
723
724
725         for (int i = 0; i < 30; i++) {
726             //double detune = 1 + (random.nextDouble()*2 - 1)*0.0001;
727             double w = 0.2;
728             double ai = aa[i];
729             if (i > 10) {
730                 w = 5;
731                 ai *= 10;
732             }
733             int adjust = 0;
734             if (i > 5) {
735                 adjust = (i - 5) * 7;
736             }
737             complexGaussianDist(data, base * (i + 1) + adjust, w, ai);
738         }
739
740         SF2Sample sample = newSimpleFFTSample(sf2, "Grand_Piano", data, base, 200);
741         SF2Layer layer = newLayer(sf2, "Grand_Piano", sample);
742         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);

```

```

929     int x = 8;
930     int fftsize = 4096 * x;
931     double[] data = new double[fftsize * 2];
932     double base = x * 25;
933     double start_w = 0.05;
934     double end_w = 0.05;
935     double start_a = 0.2;
936     double end_a = 0.0001;
937     double a = start_a;
938     double a_step = Math.pow(end_a / start_a, 1.0 / 40.0);
939     for (int i = 0; i < 40; i++) {
940         double detune = 1 + (random.nextDouble() * 2 - 1) * 0.0001;
941         double w = start_w + (end_w - start_w) * (i / 40.0);
942         complexGaussianDist(data, base * (i + 1) * detune, w, a);
943         a *= a_step;
944     }
945
946
947
948     SF2Sample sample = newSimpleFFTSample(sf2, "EPiano", data, base);
949     SF2Layer layer = newLayer(sf2, "EPiano", sample);
950     SF2Region region = layer.getRegions().get(0);
951     region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
952     region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -12000);
953     region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 0);
954     region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
955     region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
956     region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, 1200);
957     region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
958     region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, -9000);
959     region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 16000);
960     return layer;
961 }
962
963 public static SF2Layer new_epiano2(SF2Soundbank sf2) {
964     Random random = new Random(302030201);
965     int x = 8;
966     int fftsize = 4096 * x;
967     double[] data = new double[fftsize * 2];
968     double base = x * 25;
969     double start_w = 0.01;
970     double end_w = 0.05;
971     double start_a = 0.2;
972     double end_a = 0.00001;
973     double a = start_a;
974     double a_step = Math.pow(end_a / start_a, 1.0 / 40.0);
975     for (int i = 0; i < 40; i++) {
976         double detune = 1 + (random.nextDouble() * 2 - 1) * 0.0001;
977         double w = start_w + (end_w - start_w) * (i / 40.0);
978         complexGaussianDist(data, base * (i + 1) * detune, w, a);
979         a *= a_step;
980     }
981
982     SF2Sample sample = newSimpleFFTSample(sf2, "EPiano", data, base);
983     SF2Layer layer = newLayer(sf2, "EPiano", sample);
984     SF2Region region = layer.getRegions().get(0);
985     region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
986     region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -12000);
987     region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 0);
988     region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 8000);
989     region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
990     region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, 2400);

```



```

991     region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
992     region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, -9000);
993     region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 16000);
994     region.putInteger(SF2Region.GENERATOR_INITIALATTENUATION, -100);
995     return layer;
996 }
997
998 public static SF2Layer new_bass1(SF2Soundbank sf2) {
999     int x = 8;
1000     int fftsize = 4096 * x;
1001     double[] data = new double[fftsize * 2];
1002     double base = x * 25;
1003     double start_w = 0.05;
1004     double end_w = 0.05;
1005     double start_a = 0.2;
1006     double end_a = 0.02;
1007     double a = start_a;
1008     double a_step = Math.pow(end_a / start_a, 1.0 / 25.0);
1009
1010     double[] aa = new double[25];
1011     for (int i = 0; i < 25; i++) {
1012         aa[i] = a;
1013         a *= a_step;
1014     }
1015
1016     aa[0] *= 8;
1017     aa[1] *= 4;
1018     aa[3] *= 8;
1019     aa[5] *= 8;
1020
1021     for (int i = 0; i < 25; i++) {
1022         double w = start_w + (end_w - start_w) * (i / 40.0);
1023         complexGaussianDist(data, base * (i + 1), w, aa[i]);
1024     }
1025
1026
1027     SF2Sample sample = newSimpleFFTSample(sf2, "Bass", data, base);
1028     SF2Layer layer = newLayer(sf2, "Bass", sample);
1029     SF2Region region = layer.getRegions().get(0);
1030     region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
1031     region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -12000);
1032     region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 0);
1033     region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
1034     region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
1035     region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, -3000);
1036     region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
1037     region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, -5000);
1038     region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 11000);
1039     region.putInteger(SF2Region.GENERATOR_INITIALATTENUATION, -100);
1040     return layer;
1041 }
1042
1043 public static SF2Layer new_synthbass(SF2Soundbank sf2) {
1044     int x = 8;
1045     int fftsize = 4096 * x;
1046     double[] data = new double[fftsize * 2];
1047     double base = x * 25;
1048     double start_w = 0.05;
1049     double end_w = 0.05;
1050     double start_a = 0.2;
1051     double end_a = 0.02;
1052     double a = start_a;

```

```

1053     double a_step = Math.pow(end_a / start_a, 1.0 / 25.0);
1054
1055     double[] aa = new double[25];
1056     for (int i = 0; i < 25; i++) {
1057         aa[i] = a;
1058         a *= a_step;
1059     }
1060
1061     aa[0] *= 16;
1062     aa[1] *= 4;
1063     aa[3] *= 16;
1064     aa[5] *= 8;
1065
1066     for (int i = 0; i < 25; i++) {
1067         double w = start_w + (end_w - start_w) * (i / 40.0);
1068         complexGaussianDist(data, base * (i + 1), w, aa[i]);
1069     }
1070
1071
1072     SF2Sample sample = newSimpleFFTSample(sf2, "Bass", data, base);
1073     SF2Layer layer = newLayer(sf2, "Bass", sample);
1074     SF2Region region = layer.getRegions().get(0);
1075     region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
1076     region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -12000);
1077     region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 0);
1078     region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
1079     region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
1080     region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, -3000);
1081     region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
1082     region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, -3000);
1083     region.putInteger(SF2Region.GENERATOR_INITIALFILTERQ, 100);
1084     region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 8000);
1085     region.putInteger(SF2Region.GENERATOR_INITIALATTENUATION, -100);
1086     return layer;
1087 }
1088
1089 public static SF2Layer new_bass2(SF2Soundbank sf2) {
1090     int x = 8;
1091     int fftsize = 4096 * x;
1092     double[] data = new double[fftsize * 2];
1093     double base = x * 25;
1094     double start_w = 0.05;
1095     double end_w = 0.05;
1096     double start_a = 0.2;
1097     double end_a = 0.002;
1098     double a = start_a;
1099     double a_step = Math.pow(end_a / start_a, 1.0 / 25.0);
1100
1101     double[] aa = new double[25];
1102     for (int i = 0; i < 25; i++) {
1103         aa[i] = a;
1104         a *= a_step;
1105     }
1106
1107     aa[0] *= 8;
1108     aa[1] *= 4;
1109     aa[3] *= 8;
1110     aa[5] *= 8;
1111
1112     for (int i = 0; i < 25; i++) {
1113         double w = start_w + (end_w - start_w) * (i / 40.0);
1114         complexGaussianDist(data, base * (i + 1), w, aa[i]);

```

```

1115     }
1116
1117
1118     SF2Sample sample = newSimpleFFTSample(sf2, "Bass2", data, base);
1119     SF2Layer layer = newLayer(sf2, "Bass2", sample);
1120     SF2Region region = layer.getRegions().get(0);
1121     region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
1122     region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -8000);
1123     region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 0);
1124     region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
1125     region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
1126     region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, -6000);
1127     region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
1128     region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 5000);
1129     region.putInteger(SF2Region.GENERATOR_INITIALATTENUATION, -100);
1130     return layer;
1131 }
1132
1133 public static SF2Layer new_solostring(SF2Soundbank sf2) {
1134     int x = 8;
1135     int fftsize = 4096 * x;
1136     double[] data = new double[fftsize * 2];
1137     double base = x * 25;
1138     double start_w = 2;
1139     double end_w = 2;
1140     double start_a = 0.2;
1141     double end_a = 0.01;
1142
1143     double[] aa = new double[18];
1144     double a = start_a;
1145     double a_step = Math.pow(end_a / start_a, 1.0 / 40.0);
1146     for (int i = 0; i < aa.length; i++) {
1147         a *= a_step;
1148         aa[i] = a;
1149     }
1150
1151     aa[0] *= 5;
1152     aa[1] *= 5;
1153     aa[2] *= 5;
1154     aa[3] *= 4;
1155     aa[4] *= 4;
1156     aa[5] *= 3;
1157     aa[6] *= 3;
1158     aa[7] *= 2;
1159
1160     for (int i = 0; i < aa.length; i++) {
1161         double w = start_w + (end_w - start_w) * (i / 40.0);
1162         complexGaussianDist(data, base * (i + 1), w, a);
1163     }
1164     SF2Sample sample = newSimpleFFTSample(sf2, "Strings", data, base);
1165     SF2Layer layer = newLayer(sf2, "Strings", sample);
1166     SF2Region region = layer.getRegions().get(0);
1167     region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
1168     region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -5000);
1169     region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 1000);
1170     region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
1171     region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, -100);
1172     region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 9500);
1173     region.putInteger(SF2Region.GENERATOR_FREQVIBLFO, -1000);
1174     region.putInteger(SF2Region.GENERATOR_VIBLFOTOPITCH, 15);
1175     return layer;
1176

```

```

1177     }
1178
1179     public static SF2Layer new_orchhit(SF2Soundbank sf2) {
1180         int x = 8;
1181         int fftsize = 4096 * x;
1182         double[] data = new double[fftsize * 2];
1183         double base = x * 25;
1184         double start_w = 2;
1185         double end_w = 80;
1186         double start_a = 0.2;
1187         double end_a = 0.001;
1188         double a = start_a;
1189         double a_step = Math.pow(end_a / start_a, 1.0 / 40.0);
1190         for (int i = 0; i < 40; i++) {
1191             double w = start_w + (end_w - start_w) * (i / 40.0);
1192             complexGaussianDist(data, base * (i + 1), w, a);
1193             a *= a_step;
1194         }
1195         complexGaussianDist(data, base * 4, 300, 1);
1196
1197         SF2Sample sample = newSimpleFFTSample(sf2, "Och_Strings", data, base);
1198         SF2Layer layer = newLayer(sf2, "Och_Strings", sample);
1199         SF2Region region = layer.getRegions().get(0);
1200         region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
1201         region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -5000);
1202         region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 200);
1203         region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 200);
1204         region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
1205         region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 9500);
1206         return layer;
1207     }
1208
1209 }
1210
1211     public static SF2Layer new_string2(SF2Soundbank sf2) {
1212         int x = 8;
1213         int fftsize = 4096 * x;
1214         double[] data = new double[fftsize * 2];
1215         double base = x * 25;
1216         double start_w = 2;
1217         double end_w = 80;
1218         double start_a = 0.2;
1219         double end_a = 0.001;
1220         double a = start_a;
1221         double a_step = Math.pow(end_a / start_a, 1.0 / 40.0);
1222         for (int i = 0; i < 40; i++) {
1223             double w = start_w + (end_w - start_w) * (i / 40.0);
1224             complexGaussianDist(data, base * (i + 1), w, a);
1225             a *= a_step;
1226         }
1227         SF2Sample sample = newSimpleFFTSample(sf2, "Strings", data, base);
1228         SF2Layer layer = newLayer(sf2, "Strings", sample);
1229         SF2Region region = layer.getRegions().get(0);
1230         region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
1231         region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -5000);
1232         region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 1000);
1233         region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
1234         region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, -100);
1235         region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 9500);
1236         return layer;
1237     }
1238 }

```

```

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);

```

```

1301     region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
1302     region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -6000);
1303     region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, -1000);
1304     region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
1305     region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, -100);
1306     region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 9500);
1307     return layer;
1308
1309 }
1310
1311 public static SF2Layer new_ch_organ(SF2Soundbank sf2) {
1312     int x = 1;
1313     int fftsize = 4096 * x;
1314     double[] data = new double[fftsize * 2];
1315     double base = x * 15;
1316     double start_w = 0.01;
1317     double end_w = 0.01;
1318     double start_a = 0.2;
1319     double end_a = 0.001;
1320     double a = start_a;
1321     double a_step = Math.pow(end_a / start_a, 1.0 / 60.0);
1322
1323     double[] aa = new double[60];
1324     for (int i = 0; i < aa.length; i++) {
1325         a *= a_step;
1326         aa[i] = a;
1327     }
1328
1329     aa[0] *= 5;
1330     aa[1] *= 2;
1331     aa[2] = 0;
1332     aa[4] = 0;
1333     aa[5] = 0;
1334     aa[7] *= 7;
1335     aa[9] = 0;
1336     aa[10] = 0;
1337     aa[12] = 0;
1338     aa[15] *= 7;
1339     aa[18] = 0;
1340     aa[20] = 0;
1341     aa[24] = 0;
1342     aa[27] *= 5;
1343     aa[29] = 0;
1344     aa[30] = 0;
1345     aa[33] = 0;
1346     aa[36] *= 4;
1347     aa[37] = 0;
1348     aa[39] = 0;
1349     aa[42] = 0;
1350     aa[43] = 0;
1351     aa[47] = 0;
1352     aa[50] *= 4;
1353     aa[52] = 0;
1354     aa[55] = 0;
1355     aa[57] = 0;
1356
1357
1358     aa[10] *= 0.1;
1359     aa[11] *= 0.1;
1360     aa[12] *= 0.1;
1361     aa[13] *= 0.1;
1362

```

```

1363 aa[17] *= 0.1;
1364 aa[18] *= 0.1;
1365 aa[19] *= 0.1;
1366 aa[20] *= 0.1;
1367
1368 for (int i = 0; i < 60; i++) {
1369     double w = start_w + (end_w - start_w) * (i / 40.0);
1370     complexGaussianDist(data, base * (i + 1), w, aa[i]);
1371     a *= a_step;
1372 }
1373 SF2Sample sample = newSimpleFFTSample(sf2, "Organ", data, base);
1374 SF2Layer layer = newLayer(sf2, "Organ", sample);
1375 SF2Region region = layer.getRegions().get(0);
1376 region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
1377 region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -10000);
1378 region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, -1000);
1379 return layer;
1380
1381 }
1382
1383 public static SF2Layer new_flute(SF2Soundbank sf2) {
1384     int x = 8;
1385     int fftsize = 4096 * x;
1386     double[] data = new double[fftsize * 2];
1387     double base = x * 15;
1388
1389     complexGaussianDist(data, base * 1, 0.001, 0.5);
1390     complexGaussianDist(data, base * 2, 0.001, 0.5);
1391     complexGaussianDist(data, base * 3, 0.001, 0.5);
1392     complexGaussianDist(data, base * 4, 0.01, 0.5);
1393
1394     complexGaussianDist(data, base * 4, 100, 120);
1395     complexGaussianDist(data, base * 6, 100, 40);
1396     complexGaussianDist(data, base * 8, 100, 80);
1397
1398     complexGaussianDist(data, base * 5, 0.001, 0.05);
1399     complexGaussianDist(data, base * 6, 0.001, 0.06);
1400     complexGaussianDist(data, base * 7, 0.001, 0.04);
1401     complexGaussianDist(data, base * 8, 0.005, 0.06);
1402     complexGaussianDist(data, base * 9, 0.005, 0.06);
1403     complexGaussianDist(data, base * 10, 0.01, 0.1);
1404     complexGaussianDist(data, base * 11, 0.08, 0.7);
1405     complexGaussianDist(data, base * 12, 0.08, 0.6);
1406     complexGaussianDist(data, base * 13, 0.08, 0.6);
1407     complexGaussianDist(data, base * 14, 0.08, 0.6);
1408     complexGaussianDist(data, base * 15, 0.08, 0.5);
1409     complexGaussianDist(data, base * 16, 0.08, 0.5);
1410     complexGaussianDist(data, base * 17, 0.08, 0.2);
1411
1412
1413     complexGaussianDist(data, base * 1, 10, 8);
1414     complexGaussianDist(data, base * 2, 10, 8);
1415     complexGaussianDist(data, base * 3, 10, 8);
1416     complexGaussianDist(data, base * 4, 10, 8);
1417     complexGaussianDist(data, base * 5, 10, 8);
1418     complexGaussianDist(data, base * 6, 20, 9);
1419     complexGaussianDist(data, base * 7, 20, 9);
1420     complexGaussianDist(data, base * 8, 20, 9);
1421     complexGaussianDist(data, base * 9, 20, 8);
1422     complexGaussianDist(data, base * 10, 30, 8);
1423     complexGaussianDist(data, base * 11, 30, 9);
1424     complexGaussianDist(data, base * 12, 30, 9);

```

```

1425     complexGaussianDist(data, base * 13, 30, 8);
1426     complexGaussianDist(data, base * 14, 30, 8);
1427     complexGaussianDist(data, base * 15, 30, 7);
1428     complexGaussianDist(data, base * 16, 30, 7);
1429     complexGaussianDist(data, base * 17, 30, 6);
1430
1431     SF2Sample sample = newSimpleFFTSample(sf2, "Flute", data, base);
1432     SF2Layer layer = newLayer(sf2, "Flute", sample);
1433     SF2Region region = layer.getRegions().get(0);
1434     region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
1435     region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -6000);
1436     region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, -1000);
1437     region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
1438     region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, -100);
1439     region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 9500);
1440     return layer;
1441
1442 }
1443
1444 public static SF2Layer new_horn(SF2Soundbank sf2) {
1445     int x = 8;
1446     int fftsize = 4096 * x;
1447     double[] data = new double[fftsize * 2];
1448     double base = x * 15;
1449
1450     double start_a = 0.5;
1451     double end_a = 0.000000000001;
1452     double a = start_a;
1453     double a_step = Math.pow(end_a / start_a, 1.0 / 40.0);
1454     for (int i = 0; i < 40; i++) {
1455         if (i == 0)
1456             complexGaussianDist(data, base * (i + 1), 0.1, a * 0.2);
1457         else
1458             complexGaussianDist(data, base * (i + 1), 0.1, a);
1459         a *= a_step;
1460     }
1461
1462     complexGaussianDist(data, base * 2, 100, 1);
1463
1464
1465     SF2Sample sample = newSimpleFFTSample(sf2, "Horn", data, base);
1466     SF2Layer layer = newLayer(sf2, "Horn", sample);
1467     SF2Region region = layer.getRegions().get(0);
1468     region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
1469     region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -6000);
1470     region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, -1000);
1471     region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
1472     region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, -100);
1473
1474     region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, -500);
1475     region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
1476     region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, 5000);
1477     region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 4500);
1478     return layer;
1479
1480 }
1481
1482 public static SF2Layer new_trumpet(SF2Soundbank sf2) {
1483     int x = 8;
1484     int fftsize = 4096 * x;
1485     double[] data = new double[fftsize * 2];
1486     double base = x * 15;

```



```

1487
1488     double start_a = 0.5;
1489     double end_a = 0.00001;
1490     double a = start_a;
1491     double a_step = Math.pow(end_a / start_a, 1.0 / 80.0);
1492     double[] aa = new double[80];
1493     for (int i = 0; i < 80; i++) {
1494         aa[i] = a;
1495         a *= a_step;
1496     }
1497
1498     aa[0] *= 0.05;
1499     aa[1] *= 0.2;
1500     aa[2] *= 0.5;
1501     aa[3] *= 0.85;
1502
1503     for (int i = 0; i < 80; i++) {
1504         complexGaussianDist(data, base * (i + 1), 0.1, aa[i]);
1505     }
1506
1507     complexGaussianDist(data, base * 5, 300, 3);
1508
1509
1510     SF2Sample sample = newSimpleFFTSample(sf2, "Trumpet", data, base);
1511     SF2Layer layer = newLayer(sf2, "Trumpet", sample);
1512     SF2Region region = layer.getRegions().get(0);
1513     region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
1514     region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -10000);
1515     region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 0);
1516     region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
1517     region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, -100);
1518
1519     region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, -4000);
1520     region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, -2500);
1521     region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, 5000);
1522     region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 4500);
1523     region.putInteger(SF2Region.GENERATOR_INITIALFILTERQ, 10);
1524     return layer;
1525 }
1526
1527
1528 public static SF2Layer new_brass_section(SF2Soundbank sf2) {
1529     int x = 8;
1530     int fftsize = 4096 * x;
1531     double[] data = new double[fftsize * 2];
1532     double base = x * 15;
1533
1534     double start_a = 0.5;
1535     double end_a = 0.005;
1536     double a = start_a;
1537     double a_step = Math.pow(end_a / start_a, 1.0 / 30.0);
1538     double[] aa = new double[30];
1539     for (int i = 0; i < 30; i++) {
1540         aa[i] = a;
1541         a *= a_step;
1542     }
1543
1544     aa[0] *= 0.8;
1545     aa[1] *= 0.9;
1546
1547     double w = 5;
1548     for (int i = 0; i < 30; i++) {

```

```

1549         complexGaussianDist(data, base * (i + 1), 0.1 * w, aa[i] * w);
1550         w += 6; //*= w_step;
1551     }
1552
1553     complexGaussianDist(data, base * 6, 300, 2);
1554
1555
1556     SF2Sample sample = newSimpleFFTSample(sf2, "Brass_Section", data, base);
1557     SF2Layer layer = newLayer(sf2, "Brass_Section", sample);
1558     SF2Region region = layer.getRegions().get(0);
1559     region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
1560     region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -9200);
1561     region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, -1000);
1562     region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
1563     region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, -100);
1564
1565     region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, -3000);
1566     region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
1567     region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, 5000);
1568     region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 4500);
1569     return layer;
1570
1571 }
1572
1573 public static SF2Layer new_trombone(SF2Soundbank sf2) {
1574     int x = 8;
1575     int fftsize = 4096 * x;
1576     double[] data = new double[fftsize * 2];
1577     double base = x * 15;
1578
1579     double start_a = 0.5;
1580     double end_a = 0.001;
1581     double a = start_a;
1582     double a_step = Math.pow(end_a / start_a, 1.0 / 80.0);
1583     double[] aa = new double[80];
1584     for (int i = 0; i < 80; i++) {
1585         aa[i] = a;
1586         a *= a_step;
1587     }
1588
1589     aa[0] *= 0.3;
1590     aa[1] *= 0.7;
1591
1592     for (int i = 0; i < 80; i++) {
1593         complexGaussianDist(data, base * (i + 1), 0.1, aa[i]);
1594     }
1595
1596     complexGaussianDist(data, base * 6, 300, 2);
1597
1598
1599     SF2Sample sample = newSimpleFFTSample(sf2, "Trombone", data, base);
1600     SF2Layer layer = newLayer(sf2, "Trombone", sample);
1601     SF2Region region = layer.getRegions().get(0);
1602     region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
1603     region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -8000);
1604     region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, -1000);
1605     region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
1606     region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, -100);
1607
1608     region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, -2000);
1609     region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
1610     region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, 5000);

```

```

1611     region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 4500);
1612     region.putInteger(SF2Region.GENERATOR_INITIALFILTERQ, 10);
1613     return layer;
1614
1615 }
1616
1617 public static SF2Layer new_sax(SF2Soundbank sf2) {
1618     int x = 8;
1619     int fftsize = 4096 * x;
1620     double[] data = new double[fftsize * 2];
1621     double base = x * 15;
1622
1623     double start_a = 0.5;
1624     double end_a = 0.01;
1625     double a = start_a;
1626     double a_step = Math.pow(end_a / start_a, 1.0 / 40.0);
1627     for (int i = 0; i < 40; i++) {
1628         if (i == 0 || i == 2)
1629             complexGaussianDist(data, base * (i + 1), 0.1, a * 4);
1630         else
1631             complexGaussianDist(data, base * (i + 1), 0.1, a);
1632         a *= a_step;
1633     }
1634
1635     complexGaussianDist(data, base * 4, 200, 1);
1636
1637     SF2Sample sample = newSimpleFFTSample(sf2, "Sax", data, base);
1638     SF2Layer layer = newLayer(sf2, "Sax", sample);
1639     SF2Region region = layer.getRegions().get(0);
1640     region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
1641     region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -6000);
1642     region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, -1000);
1643     region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
1644     region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, -100);
1645
1646     region.putInteger(SF2Region.GENERATOR_ATTACKMODENV, -3000);
1647     region.putInteger(SF2Region.GENERATOR_RELEASEMODENV, 12000);
1648     region.putInteger(SF2Region.GENERATOR_MODENVTOFILTERFC, 5000);
1649     region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 4500);
1650     return layer;
1651
1652 }
1653
1654 public static SF2Layer new_oboe(SF2Soundbank sf2) {
1655     int x = 8;
1656     int fftsize = 4096 * x;
1657     double[] data = new double[fftsize * 2];
1658     double base = x * 15;
1659
1660     complexGaussianDist(data, base * 5, 100, 80);
1661
1662
1663     complexGaussianDist(data, base * 1, 0.01, 0.53);
1664     complexGaussianDist(data, base * 2, 0.01, 0.51);
1665     complexGaussianDist(data, base * 3, 0.01, 0.48);
1666     complexGaussianDist(data, base * 4, 0.01, 0.49);
1667     complexGaussianDist(data, base * 5, 0.01, 5);
1668     complexGaussianDist(data, base * 6, 0.01, 0.51);
1669     complexGaussianDist(data, base * 7, 0.01, 0.50);
1670     complexGaussianDist(data, base * 8, 0.01, 0.59);
1671     complexGaussianDist(data, base * 9, 0.01, 0.61);
1672     complexGaussianDist(data, base * 10, 0.01, 0.52);

```

```

1673 complexGaussianDist(data, base * 11, 0.01, 0.49);
1674 complexGaussianDist(data, base * 12, 0.01, 0.51);
1675 complexGaussianDist(data, base * 13, 0.01, 0.48);
1676 complexGaussianDist(data, base * 14, 0.01, 0.51);
1677 complexGaussianDist(data, base * 15, 0.01, 0.46);
1678 complexGaussianDist(data, base * 16, 0.01, 0.35);
1679 complexGaussianDist(data, base * 17, 0.01, 0.20);
1680 complexGaussianDist(data, base * 18, 0.01, 0.10);
1681 complexGaussianDist(data, base * 19, 0.01, 0.5);
1682 complexGaussianDist(data, base * 20, 0.01, 0.1);
1683
1684
1685 SF2Sample sample = newSimpleFFTSample(sf2, "Oboe", data, base);
1686 SF2Layer layer = newLayer(sf2, "Oboe", sample);
1687 SF2Region region = layer.getRegions().get(0);
1688 region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
1689 region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -6000);
1690 region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, -1000);
1691 region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
1692 region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, -100);
1693 region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 9500);
1694 return layer;
1695
1696 }
1697
1698 public static SF2Layer new_bassoon(SF2Soundbank sf2) {
1699     int x = 8;
1700     int fftsize = 4096 * x;
1701     double[] data = new double[fftsize * 2];
1702     double base = x * 15;
1703
1704     complexGaussianDist(data, base * 2, 100, 40);
1705     complexGaussianDist(data, base * 4, 100, 20);
1706
1707     complexGaussianDist(data, base * 1, 0.01, 0.53);
1708     complexGaussianDist(data, base * 2, 0.01, 5);
1709     complexGaussianDist(data, base * 3, 0.01, 0.51);
1710     complexGaussianDist(data, base * 4, 0.01, 0.48);
1711     complexGaussianDist(data, base * 5, 0.01, 1.49);
1712     complexGaussianDist(data, base * 6, 0.01, 0.51);
1713     complexGaussianDist(data, base * 7, 0.01, 0.50);
1714     complexGaussianDist(data, base * 8, 0.01, 0.59);
1715     complexGaussianDist(data, base * 9, 0.01, 0.61);
1716     complexGaussianDist(data, base * 10, 0.01, 0.52);
1717     complexGaussianDist(data, base * 11, 0.01, 0.49);
1718     complexGaussianDist(data, base * 12, 0.01, 0.51);
1719     complexGaussianDist(data, base * 13, 0.01, 0.48);
1720     complexGaussianDist(data, base * 14, 0.01, 0.51);
1721     complexGaussianDist(data, base * 15, 0.01, 0.46);
1722     complexGaussianDist(data, base * 16, 0.01, 0.35);
1723     complexGaussianDist(data, base * 17, 0.01, 0.20);
1724     complexGaussianDist(data, base * 18, 0.01, 0.10);
1725     complexGaussianDist(data, base * 19, 0.01, 0.5);
1726     complexGaussianDist(data, base * 20, 0.01, 0.1);
1727
1728
1729     SF2Sample sample = newSimpleFFTSample(sf2, "Flute", data, base);
1730     SF2Layer layer = newLayer(sf2, "Flute", sample);
1731     SF2Region region = layer.getRegions().get(0);
1732     region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
1733     region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -6000);
1734     region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, -1000);

```

```

1735     region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);
1736     region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, -100);
1737     region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 9500);
1738     return layer;
1739
1740 }
1741
1742 public static SF2Layer new_clarinet(SF2Soundbank sf2) {
1743     int x = 8;
1744     int fftsize = 4096 * x;
1745     double[] data = new double[fftsize * 2];
1746     double base = x * 15;
1747
1748     complexGaussianDist(data, base * 1, 0.001, 0.5);
1749     complexGaussianDist(data, base * 2, 0.001, 0.02);
1750     complexGaussianDist(data, base * 3, 0.001, 0.2);
1751     complexGaussianDist(data, base * 4, 0.01, 0.1);
1752
1753     complexGaussianDist(data, base * 4, 100, 60);
1754     complexGaussianDist(data, base * 6, 100, 20);
1755     complexGaussianDist(data, base * 8, 100, 20);
1756
1757     complexGaussianDist(data, base * 5, 0.001, 0.1);
1758     complexGaussianDist(data, base * 6, 0.001, 0.09);
1759     complexGaussianDist(data, base * 7, 0.001, 0.02);
1760     complexGaussianDist(data, base * 8, 0.005, 0.16);
1761     complexGaussianDist(data, base * 9, 0.005, 0.96);
1762     complexGaussianDist(data, base * 10, 0.01, 0.9);
1763     complexGaussianDist(data, base * 11, 0.08, 1.2);
1764     complexGaussianDist(data, base * 12, 0.08, 1.8);
1765     complexGaussianDist(data, base * 13, 0.08, 1.6);
1766     complexGaussianDist(data, base * 14, 0.08, 1.2);
1767     complexGaussianDist(data, base * 15, 0.08, 0.9);
1768     complexGaussianDist(data, base * 16, 0.08, 0.5);
1769     complexGaussianDist(data, base * 17, 0.08, 0.2);
1770
1771
1772     complexGaussianDist(data, base * 1, 10, 8);
1773     complexGaussianDist(data, base * 2, 10, 8);
1774     complexGaussianDist(data, base * 3, 10, 8);
1775     complexGaussianDist(data, base * 4, 10, 8);
1776     complexGaussianDist(data, base * 5, 10, 8);
1777     complexGaussianDist(data, base * 6, 20, 9);
1778     complexGaussianDist(data, base * 7, 20, 9);
1779     complexGaussianDist(data, base * 8, 20, 9);
1780     complexGaussianDist(data, base * 9, 20, 8);
1781     complexGaussianDist(data, base * 10, 30, 8);
1782     complexGaussianDist(data, base * 11, 30, 9);
1783     complexGaussianDist(data, base * 12, 30, 9);
1784     complexGaussianDist(data, base * 13, 30, 8);
1785     complexGaussianDist(data, base * 14, 30, 8);
1786     complexGaussianDist(data, base * 15, 30, 7);
1787     complexGaussianDist(data, base * 16, 30, 7);
1788     complexGaussianDist(data, base * 17, 30, 6);
1789
1790     SF2Sample sample = newSimpleFFTSample(sf2, "Clarinet", data, base);
1791     SF2Layer layer = newLayer(sf2, "Clarinet", sample);
1792     SF2Region region = layer.getRegions().get(0);
1793     region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
1794     region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -6000);
1795     region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, -1000);
1796     region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 4000);

```

```

1797     region.putInteger(SF2Region.GENERATOR_SUSTAINVENV, -100);
1798     region.putInteger(SF2Region.GENERATOR_INITIALFILTERFC, 9500);
1799     return layer;
1800
1801 }
1802
1803 public static SF2Layer new_timpani(SF2Soundbank sf2) {
1804
1805     double datab[];
1806     double datah[];
1807
1808     // Make Bass Part
1809     {
1810         int fftlen = 4096 * 8;
1811         double[] data = new double[2 * fftlen];
1812         double base = 48;
1813         complexGaussianDist(data, base * 2, 0.2, 1);
1814         complexGaussianDist(data, base * 3, 0.2, 0.7);
1815         complexGaussianDist(data, base * 5, 10, 1);
1816         complexGaussianDist(data, base * 6, 9, 1);
1817         complexGaussianDist(data, base * 8, 15, 1);
1818         complexGaussianDist(data, base * 9, 18, 0.8);
1819         complexGaussianDist(data, base * 11, 21, 0.5);
1820         complexGaussianDist(data, base * 13, 28, 0.3);
1821         complexGaussianDist(data, base * 14, 22, 0.1);
1822         randomPhase(data, new Random(3049912));
1823         ifft(data);
1824         normalize(data, 0.5);
1825         data = realPart(data);
1826
1827         double d_len = data.length;
1828         for (int i = 0; i < data.length; i++) {
1829             double g = (1.0 - (i / d_len));
1830             data[i] *= g * g;
1831         }
1832         fadeUp(data, 40);
1833         datab = data;
1834     }
1835
1836     // Make treble part
1837     {
1838         int fftlen = 4096 * 4;
1839         double[] data = new double[2 * fftlen];
1840         Random random = new Random(3049912);
1841         for (int i = 0; i < data.length; i += 2) {
1842             data[i] = (2.0 * (random.nextDouble() - 0.5)) * 0.1;
1843         }
1844         fft(data);
1845         // Remove all negative frequency
1846         for (int i = fftlen / 2; i < data.length; i++)
1847             data[i] = 0;
1848         for (int i = 1024 * 4; i < 2048 * 4; i++)
1849             data[i] = 1.0 - (i - 4096) / 4096.0;
1850         for (int i = 0; i < 300; i++) {
1851             double g = (1.0 - (i / 300.0));
1852             data[i] *= 1.0 + 20 * g * g;
1853         }
1854         for (int i = 0; i < 24; i++)
1855             data[i] = 0;
1856         randomPhase(data, new Random(3049912));
1857         ifft(data);
1858         normalize(data, 0.9);

```

```

1859     data = realPart(data);
1860     double gain = 1.0;
1861     for (int i = 0; i < data.length; i++) {
1862         data[i] *= gain;
1863         gain *= 0.9998;
1864     }
1865     datah = data;
1866 }
1867
1868 for (int i = 0; i < datah.length; i++)
1869     datab[i] += datah[i] * 0.02;
1870
1871 normalize(datab, 0.9);
1872
1873 SF2Sample sample = newSimpleDrumSample(sf2, "Timpani", datab);
1874
1875 SF2Layer layer = new SF2Layer(sf2);
1876 layer.setName("Timpani");
1877
1878 SF2GlobalRegion global = new SF2GlobalRegion();
1879 layer.setGlobalZone(global);
1880 sf2.addResource(layer);
1881
1882 SF2LayerRegion region = new SF2LayerRegion();
1883 region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 12000);
1884 region.putInteger(SF2Region.GENERATOR_INITIALATTENUATION, -100);
1885 region.setSample(sample);
1886 layer.getRegions().add(region);
1887
1888 return layer;
1889 }
1890
1891 public static SF2Layer new_melodic_toms(SF2Soundbank sf2) {
1892
1893     double datab[];
1894     double datah[];
1895
1896     // Make Bass Part
1897     {
1898         int fftlen = 4096 * 4;
1899         double[] data = new double[2 * fftlen];
1900         complexGaussianDist(data, 30, 0.5, 1);
1901         randomPhase(data, new Random(3049912));
1902         ifft(data);
1903         normalize(data, 0.8);
1904         data = realPart(data);
1905
1906         double d_len = data.length;
1907         for (int i = 0; i < data.length; i++)
1908             data[i] *= (1.0 - (i / d_len));
1909         datab = data;
1910     }
1911
1912     // Make treble part
1913     {
1914         int fftlen = 4096 * 4;
1915         double[] data = new double[2 * fftlen];
1916         Random random = new Random(3049912);
1917         for (int i = 0; i < data.length; i += 2)
1918             data[i] = (2.0 * (random.nextDouble() - 0.5)) * 0.1;
1919         fft(data);
1920         // Remove all negative frequency

```

```

1921     for (int i = fftlen / 2; i < data.length; i++)
1922         data[i] = 0;
1923     for (int i = 1024 * 4; i < 2048 * 4; i++)
1924         data[i] = 1.0 - (i - 4096) / 4096.0;
1925     for (int i = 0; i < 200; i++) {
1926         double g = (1.0 - (i / 200.0));
1927         data[i] *= 1.0 + 20 * g * g;
1928     }
1929     for (int i = 0; i < 30; i++)
1930         data[i] = 0;
1931     randomPhase(data, new Random(3049912));
1932     ifft(data);
1933     normalize(data, 0.9);
1934     data = realPart(data);
1935     double gain = 1.0;
1936     for (int i = 0; i < data.length; i++) {
1937         data[i] *= gain;
1938         gain *= 0.9996;
1939     }
1940     datah = data;
1941 }
1942
1943 for (int i = 0; i < datah.length; i++)
1944     datab[i] += datah[i] * 0.5;
1945 for (int i = 0; i < 5; i++)
1946     datab[i] *= i / 5.0;
1947
1948 normalize(datab, 0.99);
1949
1950 SF2Sample sample = newSimpleDrumSample(sf2, "Melodic_Toms", datab);
1951 sample.setOriginalPitch(63);
1952
1953 SF2Layer layer = new SF2Layer(sf2);
1954 layer.setName("Melodic_Toms");
1955
1956 SF2GlobalRegion global = new SF2GlobalRegion();
1957 layer.setGlobalZone(global);
1958 sf2.addResource(layer);
1959
1960 SF2LayerRegion region = new SF2LayerRegion();
1961 region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 12000);
1962 //region.putInteger(SF2Region.GENERATOR_SCALETUNING, 0);
1963 region.putInteger(SF2Region.GENERATOR_INITIALATTENUATION, -100);
1964 region.setSample(sample);
1965 layer.getRegions().add(region);
1966
1967 return layer;
1968 }
1969
1970 public static SF2Layer new_reverse_cymbal(SF2Soundbank sf2) {
1971     double datah[];
1972     {
1973         int fftlen = 4096 * 4;
1974         double[] data = new double[2 * fftlen];
1975         Random random = new Random(3049912);
1976         for (int i = 0; i < data.length; i += 2)
1977             data[i] = (2.0 * (random.nextDouble() - 0.5));
1978         for (int i = fftlen / 2; i < data.length; i++)
1979             data[i] = 0;
1980         for (int i = 0; i < 100; i++)
1981             data[i] = 0;

```



```

1983     for (int i = 0; i < 512 * 2; i++) {
1984         double gain = (i / (512.0 * 2.0));
1985         data[i] = 1 - gain;
1986     }
1987     datah = data;
1988 }
1989
1990 SF2Sample sample = newSimpleFFTSample(sf2, "Reverse_Cymbal",
1991     datah, 100, 20);
1992
1993 SF2Layer layer = new SF2Layer(sf2);
1994 layer.setName("Reverse_Cymbal");
1995
1996 SF2GlobalRegion global = new SF2GlobalRegion();
1997 layer.setGlobalZone(global);
1998 sf2.addResource(layer);
1999
2000 SF2LayerRegion region = new SF2LayerRegion();
2001 region.putInteger(SF2Region.GENERATOR_ATTACKVOLENV, -200);
2002 region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, -12000);
2003 region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
2004 region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, -1000);
2005 region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
2006 region.setSample(sample);
2007 layer.getRegions().add(region);
2008
2009 return layer;
2010 }
2011
2012 public static SF2Layer new_snare_drum(SF2Soundbank sf2) {
2013
2014     double datab[];
2015     double datah[];
2016
2017     // Make Bass Part
2018     {
2019         int fftlen = 4096 * 4;
2020         double[] data = new double[2 * fftlen];
2021         complexGaussianDist(data, 24, 0.5, 1);
2022         randomPhase(data, new Random(3049912));
2023         ifft(data);
2024         normalize(data, 0.5);
2025         data = realPart(data);
2026
2027         double d_len = data.length;
2028         for (int i = 0; i < data.length; i++)
2029             data[i] *= (1.0 - (i / d_len));
2030         datab = data;
2031     }
2032
2033     // Make treble part
2034     {
2035         int fftlen = 4096 * 4;
2036         double[] data = new double[2 * fftlen];
2037         Random random = new Random(3049912);
2038         for (int i = 0; i < data.length; i += 2)
2039             data[i] = (2.0 * (random.nextDouble() - 0.5)) * 0.1;
2040         fft(data);
2041         // Remove all negative frequency
2042         for (int i = fftlen / 2; i < data.length; i++)
2043             data[i] = 0;
2044         for (int i = 1024 * 4; i < 2048 * 4; i++)

```

```

2045         data[i] = 1.0 - (i - 4096) / 4096.0;
2046     for (int i = 0; i < 300; i++) {
2047         double g = (1.0 - (i / 300.0));
2048         data[i] *= 1.0 + 20 * g * g;
2049     }
2050     for (int i = 0; i < 24; i++)
2051         data[i] = 0;
2052     randomPhase(data, new Random(3049912));
2053     ifft(data);
2054     normalize(data, 0.9);
2055     data = realPart(data);
2056     double gain = 1.0;
2057     for (int i = 0; i < data.length; i++) {
2058         data[i] *= gain;
2059         gain *= 0.9998;
2060     }
2061     datah = data;
2062 }
2063
2064 for (int i = 0; i < datah.length; i++)
2065     datab[i] += datah[i];
2066 for (int i = 0; i < 5; i++)
2067     datab[i] *= i / 5.0;
2068
2069 SF2Sample sample = newSimpleDrumSample(sf2, "Snare_Drum", datab);
2070
2071 SF2Layer layer = new SF2Layer(sf2);
2072 layer.setName("Snare_Drum");
2073
2074 SF2GlobalRegion global = new SF2GlobalRegion();
2075 layer.setGlobalZone(global);
2076 sf2.addResource(layer);
2077
2078 SF2LayerRegion region = new SF2LayerRegion();
2079 region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 12000);
2080 region.putInteger(SF2Region.GENERATOR_SCALETUNING, 0);
2081 region.putInteger(SF2Region.GENERATOR_INITIALATTENUATION, -100);
2082 region.setSample(sample);
2083 layer.getRegions().add(region);
2084
2085 return layer;
2086 }
2087
2088 public static SF2Layer new_bass_drum(SF2Soundbank sf2) {
2089
2090     double datab[];
2091     double datah[];
2092
2093     // Make Bass Part
2094     {
2095         int fftlen = 4096 * 4;
2096         double[] data = new double[2 * fftlen];
2097         complexGaussianDist(data, 1.8 * 5 + 1, 2, 1);
2098         complexGaussianDist(data, 1.8 * 9 + 1, 2, 1);
2099         randomPhase(data, new Random(3049912));
2100         ifft(data);
2101         normalize(data, 0.9);
2102         data = realPart(data);
2103         double d_len = data.length;
2104         for (int i = 0; i < data.length; i++)
2105             data[i] *= (1.0 - (i / d_len));
2106         datab = data;

```

```

2107     }
2108
2109     // Make treble part
2110     {
2111         int fftlen = 4096;
2112         double[] data = new double[2 * fftlen];
2113         Random random = new Random(3049912);
2114         for (int i = 0; i < data.length; i += 2)
2115             data[i] = (2.0 * (random.nextDouble() - 0.5)) * 0.1;
2116         fft(data);
2117         // Remove all negative frequency
2118         for (int i = fftlen / 2; i < data.length; i++)
2119             data[i] = 0;
2120         for (int i = 1024; i < 2048; i++)
2121             data[i] = 1.0 - (i - 1024) / 1024.0;
2122         for (int i = 0; i < 512; i++)
2123             data[i] = 10 * i / 512.0;
2124         for (int i = 0; i < 10; i++)
2125             data[i] = 0;
2126         randomPhase(data, new Random(3049912));
2127         ifft(data);
2128         normalize(data, 0.9);
2129         data = realPart(data);
2130         double gain = 1.0;
2131         for (int i = 0; i < data.length; i++) {
2132             data[i] *= gain;
2133             gain *= 0.999;
2134         }
2135         datah = data;
2136     }
2137
2138     for (int i = 0; i < datah.length; i++)
2139         datab[i] += datah[i] * 0.5;
2140     for (int i = 0; i < 5; i++)
2141         datab[i] *= i / 5.0;
2142
2143     SF2Sample sample = newSimpleDrumSample(sf2, "Bass_Drum", datab);
2144
2145     SF2Layer layer = new SF2Layer(sf2);
2146     layer.setName("Bass_Drum");
2147
2148     SF2GlobalRegion global = new SF2GlobalRegion();
2149     layer.setGlobalZone(global);
2150     sf2.addResource(layer);
2151
2152     SF2LayerRegion region = new SF2LayerRegion();
2153     region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 12000);
2154     region.putInteger(SF2Region.GENERATOR_SCALETUNING, 0);
2155     region.putInteger(SF2Region.GENERATOR_INITIALATTENUATION, -100);
2156     region.setSample(sample);
2157     layer.getRegions().add(region);
2158
2159     return layer;
2160 }
2161
2162 public static SF2Layer new_tom(SF2Soundbank sf2) {
2163
2164     double datab[];
2165     double datah[];
2166
2167     // Make Bass Part
2168     {

```

```

2169     int fftlen = 4096 * 4;
2170     double[] data = new double[2 * fftlen];
2171     complexGaussianDist(data, 30, 0.5, 1);
2172     randomPhase(data, new Random(3049912));
2173     ifft(data);
2174     normalize(data, 0.8);
2175     data = realPart(data);
2176
2177     double d_len = data.length;
2178     for (int i = 0; i < data.length; i++)
2179         data[i] *= (1.0 - (i / d_len));
2180     datab = data;
2181 }
2182
2183 // Make treble part
2184 {
2185     int fftlen = 4096 * 4;
2186     double[] data = new double[2 * fftlen];
2187     Random random = new Random(3049912);
2188     for (int i = 0; i < data.length; i += 2)
2189         data[i] = (2.0 * (random.nextDouble() - 0.5)) * 0.1;
2190     fft(data);
2191     // Remove all negative frequency
2192     for (int i = fftlen / 2; i < data.length; i++)
2193         data[i] = 0;
2194     for (int i = 1024 * 4; i < 2048 * 4; i++)
2195         data[i] = 1.0 - (i - 4096) / 4096.0;
2196     for (int i = 0; i < 200; i++) {
2197         double g = (1.0 - (i / 200.0));
2198         data[i] *= 1.0 + 20 * g * g;
2199     }
2200     for (int i = 0; i < 30; i++)
2201         data[i] = 0;
2202     randomPhase(data, new Random(3049912));
2203     ifft(data);
2204     normalize(data, 0.9);
2205     data = realPart(data);
2206     double gain = 1.0;
2207     for (int i = 0; i < data.length; i++) {
2208         data[i] *= gain;
2209         gain *= 0.9996;
2210     }
2211     datah = data;
2212 }
2213
2214 for (int i = 0; i < datah.length; i++)
2215     datab[i] += datah[i] * 0.5;
2216 for (int i = 0; i < 5; i++)
2217     datab[i] *= i / 5.0;
2218
2219 normalize(datab, 0.99);
2220
2221 SF2Sample sample = new SimpleDrumSample(sf2, "Tom", datab);
2222 sample.setOriginalPitch(50);
2223
2224 SF2Layer layer = new SF2Layer(sf2);
2225 layer.setName("Tom");
2226
2227 SF2GlobalRegion global = new SF2GlobalRegion();
2228 layer.setGlobalZone(global);
2229 sf2.addResource(layer);
2230

```

```

2231 SF2LayerRegion region = new SF2LayerRegion();
2232 region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 12000);
2233 //region.putInteger(SF2Region.GENERATOR_SCALETUNING, 0);
2234 region.putInteger(SF2Region.GENERATOR_INITIALATTENUATION, -100);
2235 region.setSample(sample);
2236 layer.getRegions().add(region);
2237
2238 return layer;
2239 }
2240
2241 public static SF2Layer new_closed_hihat(SF2Soundbank sf2) {
2242     double datah[];
2243
2244     // Make treble part
2245     {
2246         int fftlen = 4096 * 4;
2247         double[] data = new double[2 * fftlen];
2248         Random random = new Random(3049912);
2249         for (int i = 0; i < data.length; i += 2)
2250             data[i] = (2.0 * (random.nextDouble() - 0.5)) * 0.1;
2251         fft(data);
2252         // Remove all negative frequency
2253         for (int i = fftlen / 2; i < data.length; i++)
2254             data[i] = 0;
2255         for (int i = 1024 * 4; i < 2048 * 4; i++)
2256             data[i] = 1.0 - (i - 4096) / 4096.0;
2257         for (int i = 0; i < 2048; i++)
2258             data[i] = 0.2 + 0.8 * (i / 2048.0);
2259         randomPhase(data, new Random(3049912));
2260         ifft(data);
2261         normalize(data, 0.9);
2262         data = realPart(data);
2263         double gain = 1.0;
2264         for (int i = 0; i < data.length; i++) {
2265             data[i] *= gain;
2266             gain *= 0.9996;
2267         }
2268         datah = data;
2269     }
2270
2271     for (int i = 0; i < 5; i++)
2272         datah[i] *= i / 5.0;
2273     SF2Sample sample = newSimpleDrumSample(sf2, "Closed_Hi-Hat", datah);
2274
2275     SF2Layer layer = new SF2Layer(sf2);
2276     layer.setName("Closed_Hi-Hat");
2277
2278     SF2GlobalRegion global = new SF2GlobalRegion();
2279     layer.setGlobalZone(global);
2280     sf2.addResource(layer);
2281
2282     SF2LayerRegion region = new SF2LayerRegion();
2283     region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 12000);
2284     region.putInteger(SF2Region.GENERATOR_SCALETUNING, 0);
2285     region.putInteger(SF2Region.GENERATOR_EXCLUSIVECLASS, 1);
2286     region.setSample(sample);
2287     layer.getRegions().add(region);
2288
2289     return layer;
2290 }
2291
2292 public static SF2Layer new_open_hihat(SF2Soundbank sf2) {

```

```

2293 double datah[];
2294 {
2295     int fftlen = 4096 * 4;
2296     double[] data = new double[2 * fftlen];
2297     Random random = new Random(3049912);
2298     for (int i = 0; i < data.length; i += 2)
2299         data[i] = (2.0 * (random.nextDouble() - 0.5));
2300     for (int i = fftlen / 2; i < data.length; i++)
2301         data[i] = 0;
2302     for (int i = 0; i < 200; i++)
2303         data[i] = 0;
2304     for (int i = 0; i < 2048 * 4; i++) {
2305         double gain = (i / (2048.0 * 4.0));
2306         data[i] = gain;
2307     }
2308     datah = data;
2309 }
2310
2311 SF2Sample sample = newSimpleFFTSample(sf2, "Open_Hi-Hat", datah, 1000, 5);
2312
2313 SF2Layer layer = new SF2Layer(sf2);
2314 layer.setName("Open_Hi-Hat");
2315
2316 SF2GlobalRegion global = new SF2GlobalRegion();
2317 layer.setGlobalZone(global);
2318 sf2.addResource(layer);
2319
2320 SF2LayerRegion region = new SF2LayerRegion();
2321 region.putInteger(SF2Region.GENERATOR_DECAYVOLENV, 1500);
2322 region.putInteger(SF2Region.GENERATOR_SAMPLEMODES, 1);
2323 region.putInteger(SF2Region.GENERATOR_RELEASEVOLENV, 1500);
2324 region.putInteger(SF2Region.GENERATOR_SUSTAINVOLENV, 1000);
2325 region.putInteger(SF2Region.GENERATOR_SCALETUNING, 0);
2326 region.putInteger(SF2Region.GENERATOR_EXCLUSIVECLASS, 1);
2327 region.setSample(sample);
2328 layer.getRegions().add(region);
2329
2330 return layer;
2331 }
2332
2333 public static SF2Layer new_crash_cymbal(SF2Soundbank sf2) {
2334     double datah[];
2335     {
2336         int fftlen = 4096 * 4;
2337         double[] data = new double[2 * fftlen];
2338         Random random = new Random(3049912);
2339         for (int i = 0; i < data.length; i += 2)
2340             data[i] = (2.0 * (random.nextDouble() - 0.5));
2341         for (int i = fftlen / 2; i < data.length; i++)
2342             data[i] = 0;
2343         for (int i = 0; i < 100; i++)
2344             data[i] = 0;
2345         for (int i = 0; i < 512 * 2; i++) {
2346             double gain = (i / (512.0 * 2.0));
2347             data[i] = gain;
2348         }
2349         datah = data;
2350     }
2351
2352     SF2Sample sample = newSimpleFFTSample(sf2, "Crash_Cymbal", datah, 1000, 5);
2353
2354     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);

```

```

2417     sf2.addResource(layer);
2418
2419     SF2LayerRegion region = new SF2LayerRegion();
2420     region.putInteger(SF2Region.GENERATOR_RELEASEVENV, 12000);
2421     region.putInteger(SF2Region.GENERATOR_SCALETUNING, 0);
2422     region.putInteger(SF2Region.GENERATOR_INITIALATTENUATION, -50);
2423     region.setSample(sample);
2424     layer.getRegions().add(region);
2425
2426     return layer;
2427
2428 }
2429
2430 public static SF2Sample newSimpleFFTSample(SF2Soundbank sf2, String name,
2431     double[] data, double base) {
2432     return newSimpleFFTSample(sf2, name, data, base, 10);
2433 }
2434
2435 public static SF2Sample newSimpleFFTSample(SF2Soundbank sf2, String name,
2436     double[] data, double base, int fadeuptime) {
2437
2438     int fftsize = data.length / 2;
2439     AudioFormat format = new AudioFormat(44100, 16, 1, true, false);
2440     double basefreq = (base / fftsize) * format.getSampleRate() * 0.5;
2441
2442     randomPhase(data);
2443     ifft(data);
2444     data = realPart(data);
2445     normalize(data, 0.9);
2446     float[] fdata = toFloat(data);
2447     fdata = loopExtend(fdata, fdata.length + 512);
2448     fadeUp(fdata, fadeuptime);
2449     byte[] bdata = toBytes(fdata, format);
2450
2451     /*
2452      * Create SoundFont2 sample.
2453      */
2454     SF2Sample sample = new SF2Sample(sf2);
2455     sample.setName(name);
2456     sample.setData(bdata);
2457     sample.setStartLoop(256);
2458     sample.setEndLoop(fftsize + 256);
2459     sample.setSampleRate((long) format.getSampleRate());
2460     double orgnote = (69 + 12)
2461         + (12 * Math.log(basefreq / 440.0) / Math.log(2));
2462     sample.setOriginalPitch((int) orgnote);
2463     sample.setPitchCorrection((byte) (-(orgnote - (int) orgnote) * 100.0));
2464     sf2.addResource(sample);
2465
2466     return sample;
2467 }
2468
2469 public static SF2Sample newSimpleFFTSample_dist(SF2Soundbank sf2,
2470     String name, double[] data, double base, double preamp) {
2471
2472     int fftsize = data.length / 2;
2473     AudioFormat format = new AudioFormat(44100, 16, 1, true, false);
2474     double basefreq = (base / fftsize) * format.getSampleRate() * 0.5;
2475
2476     randomPhase(data);
2477     ifft(data);
2478     data = realPart(data);

```


2479

2480

2481

2482

2483

2484

2485

2486

2487

2488

2489

2490

2491

2492

2493

2494

2495

2496

2497

2498

2499

2500

2501

2502

2503

2504

2505

2506

2507

2508

2509

2510

2511

2512

2513

2514

2515

2516

2517

2518

2519

2520

2521

2522

2523

2524

2525

2526

2527

2528

2529

2530

2531

2532

2533

2534

2535

2536

2537

2538

2539

2540

```

    for (int i = 0; i < data.length; i++) {
        data[i] = (1 - Math.exp(-Math.abs(data[i] * preamp)))
            * Math.signum(data[i]);
    }

    normalize(data, 0.9);
    float[] fdata = toFloat(data);
    fdata = loopExtend(fdata, fdata.length + 512);
    fadeUp(fdata, 80);
    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 newSimpleDrumSample(SF2Soundbank sf2, String name,
    double[] data) {

    int fftsize = data.length;
    AudioFormat format = new AudioFormat(44100, 16, 1, true, false);

    byte[] bdata = toBytes(toFloat(realPart(data)), 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());
    sample.setOriginalPitch(60);
    sf2.addResource(sample);

    return sample;
}

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

    SF2Layer layer = new SF2Layer(sf2);
    layer.setName(name);
    layer.getRegions().add(region);
    sf2.addResource(layer);
}

```

```

2541     return layer;
2542 }
2543
2544 public static SF2Instrument newInstrument(SF2Soundbank sf2, String name,
2545     Patch patch, SF2Layer... layers) {
2546
2547     /*
2548     * Create SoundFont2 instrument.
2549     */
2550     SF2Instrument ins = new SF2Instrument(sf2);
2551     ins.setPatch(patch);
2552     ins.setName(name);
2553     sf2.addInstrument(ins);
2554
2555     /*
2556     * Create region for instrument.
2557     */
2558     for (int i = 0; i < layers.length; i++) {
2559         SF2InstrumentRegion insregion = new SF2InstrumentRegion();
2560         insregion.setLayer(layers[i]);
2561         ins.getRegions().add(insregion);
2562     }
2563
2564     return ins;
2565 }
2566
2567 static public void ifft(double[] data) {
2568     new FFT(data.length / 2, 1).transform(data);
2569 }
2570
2571 static public void fft(double[] data) {
2572     new FFT(data.length / 2, -1).transform(data);
2573 }
2574
2575 public static void complexGaussianDist(double[] cdata, double m,
2576     double s, double v) {
2577     for (int x = 0; x < cdata.length / 4; x++) {
2578         cdata[x * 2] += v * (1.0 / (s * Math.sqrt(2 * Math.PI)))
2579             * Math.exp((-1.0 / 2.0) * Math.pow((x - m) / s, 2.0));
2580     }
2581 }
2582
2583 static public void randomPhase(double[] data) {
2584     for (int i = 0; i < data.length; i += 2) {
2585         double phase = Math.random() * 2 * Math.PI;
2586         double d = data[i];
2587         data[i] = Math.sin(phase) * d;
2588         data[i + 1] = Math.cos(phase) * d;
2589     }
2590 }
2591
2592 static public void randomPhase(double[] data, Random random) {
2593     for (int i = 0; i < data.length; i += 2) {
2594         double phase = random.nextDouble() * 2 * Math.PI;
2595         double d = data[i];
2596         data[i] = Math.sin(phase) * d;
2597         data[i + 1] = Math.cos(phase) * d;
2598     }
2599 }
2600
2601 static public void normalize(double[] data, double target) {
2602     double maxvalue = 0;

```

```

2603     for (int i = 0; i < data.length; i++) {
2604         if (data[i] > maxvalue)
2605             maxvalue = data[i];
2606         if (-data[i] > maxvalue)
2607             maxvalue = -data[i];
2608     }
2609     if (maxvalue == 0)
2610         return;
2611     double gain = target / maxvalue;
2612     for (int i = 0; i < data.length; i++)
2613         data[i] *= gain;
2614 }
2615
2616 static public void normalize(float[] data, double target) {
2617     double maxvalue = 0.5;
2618     for (int i = 0; i < data.length; i++) {
2619         if (data[i * 2] > maxvalue)
2620             maxvalue = data[i * 2];
2621         if (-data[i * 2] > maxvalue)
2622             maxvalue = -data[i * 2];
2623     }
2624     double gain = target / maxvalue;
2625     for (int i = 0; i < data.length; i++)
2626         data[i * 2] *= gain;
2627 }
2628
2629 static public double[] realPart(double[] in) {
2630     double[] out = new double[in.length / 2];
2631     for (int i = 0; i < out.length; i++) {
2632         out[i] = in[i * 2];
2633     }
2634     return out;
2635 }
2636
2637 static public double[] imgPart(double[] in) {
2638     double[] out = new double[in.length / 2];
2639     for (int i = 0; i < out.length; i++) {
2640         out[i] = in[i * 2];
2641     }
2642     return out;
2643 }
2644
2645 static public float[] toFloat(double[] in) {
2646     float[] out = new float[in.length];
2647     for (int i = 0; i < out.length; i++) {
2648         out[i] = (float) in[i];
2649     }
2650     return out;
2651 }
2652
2653 static public byte[] toBytes(float[] in, AudioFormat format) {
2654     byte[] out = new byte[in.length * format.getFrameSize()];
2655     return AudioFloatConverter.getConverter(format).toByteArray(in, out);
2656 }
2657
2658 static public void fadeUp(double[] data, int samples) {
2659     double dsamples = samples;
2660     for (int i = 0; i < samples; i++)
2661         data[i] *= i / dsamples;
2662 }
2663
2664 static public void fadeUp(float[] data, int samples) {

```

```

2665     double dsamples = samples;
2666     for (int i = 0; i < samples; i++)
2667         data[i] *= i / dsamples;
2668 }
2669
2670 static public double[] loopExtend(double[] data, int newsize) {
2671     double[] outdata = new double[newsize];
2672     int p_len = data.length;
2673     int p_ps = 0;
2674     for (int i = 0; i < outdata.length; i++) {
2675         outdata[i] = data[p_ps];
2676         p_ps++;
2677         if (p_ps == p_len)
2678             p_ps = 0;
2679     }
2680     return outdata;
2681 }
2682
2683 static public float[] loopExtend(float[] data, int newsize) {
2684     float[] outdata = new float[newsize];
2685     int p_len = data.length;
2686     int p_ps = 0;
2687     for (int i = 0; i < outdata.length; i++) {
2688         outdata[i] = data[p_ps];
2689         p_ps++;
2690         if (p_ps == p_len)
2691             p_ps = 0;
2692     }
2693     return outdata;
2694 }
2695 }

```

32 com/sun/media/sound/FFT.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 /**
28  * Fast Fourier Transformer.
29  *
30  * @author Karl Helgason
31  */
32 public final class FFT {
33
34     private double[] w;
35     private int fftFrameSize;
36     private int sign;
37     private int[] bitm_array;
38     private int fftFrameSize2;
39
40     // Sign = -1 is FFT, 1 is IFFT (inverse FFT)
41     // Data = Interlaced double array to be transformed.
42     // The order is: real (sin), complex (cos)
43     // Framesize must be power of 2
44     public FFT(int fftFrameSize, int sign) {
45         w = computeTwiddleFactors(fftFrameSize, sign);
46
47         this.fftFrameSize = fftFrameSize;
48         this.sign = sign;
49         fftFrameSize2 = fftFrameSize << 1;
50
51         // Pre-process Bit-Reversal
52         bitm_array = new int[fftFrameSize2];
53         for (int i = 2; i < fftFrameSize2; i += 2) {
54             int j;
55             int bitm;
56             for (bitm = 2, j = 0; bitm < fftFrameSize2; bitm <= 1) {
57                 if ((i & bitm) != 0)
58                     j++;
59                 j <= 1;
60             }
61         }
62     }
63 }
```

```

61         bitm_array[i] = j;
62     }
63
64 }
65
66 public void transform(double[] data) {
67     bitreversal(data);
68     calc(fftFrameSize, data, sign, w);
69 }
70
71 private final static double[] computeTwiddleFactors(int fftFrameSize,
72     int sign) {
73
74     int imax = (int) (Math.log(fftFrameSize) / Math.log(2.));
75
76     double[] warray = new double[(fftFrameSize - 1) * 4];
77     int w_index = 0;
78
79     for (int i = 0, nstep = 2; i < imax; i++) {
80         int jmax = nstep;
81         nstep <= 1;
82
83         double wr = 1.0;
84         double wi = 0.0;
85
86         double arg = Math.PI / (jmax >> 1);
87         double wfr = Math.cos(arg);
88         double wfi = sign * Math.sin(arg);
89
90         for (int j = 0; j < jmax; j += 2) {
91             warray[w_index++] = wr;
92             warray[w_index++] = wi;
93
94             double tempr = wr;
95             wr = tempr * wfr - wi * wfi;
96             wi = tempr * wfi + wi * wfr;
97         }
98     }
99
100     // PRECOMPUTATION of wwr1, wwi1 for factor 4 Decomposition (3 * complex
101     // operators and 8 +/- complex operators)
102     {
103         w_index = 0;
104         int w_index2 = warray.length >> 1;
105         for (int i = 0, nstep = 2; i < (imax - 1); i++) {
106             int jmax = nstep;
107             nstep *= 2;
108
109             int ii = w_index + jmax;
110             for (int j = 0; j < jmax; j += 2) {
111                 double wr = warray[w_index++];
112                 double wi = warray[w_index++];
113                 double wr1 = warray[ii++];
114                 double wi1 = warray[ii++];
115                 warray[w_index2++] = wr * wr1 - wi * wi1;
116                 warray[w_index2++] = wr * wi1 + wi * wr1;
117             }
118         }
119     }
120 }
121
122 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);

```

```

185         return;
186     }
187     nstep <= 2;
188     int ii = i + jmax;
189     int iii = i + w_len;
190
191     {
192         i += 2;
193         ii += 2;
194         iii += 2;
195
196         for (int n = 0; n < fftFrameSize2; n += nstep) {
197             int m = n + jmax;
198
199             double datam1_r = data[m];
200             double datam1_i = data[m + 1];
201             double datan1_r = data[n];
202             double datan1_i = data[n + 1];
203
204             n += nnstep;
205             m += nnstep;
206             double datam2_r = data[m];
207             double datam2_i = data[m + 1];
208             double datan2_r = data[n];
209             double datan2_i = data[n + 1];
210
211             double tempr = datam1_r;
212             double tempi = datam1_i;
213
214             datam1_r = datan1_r - tempr;
215             datam1_i = datan1_i - tempi;
216             datan1_r = datan1_r + tempr;
217             datan1_i = datan1_i + tempi;
218
219             double n2w1r = datan2_r;
220             double n2w1i = datan2_i;
221             double m2ww1r = datam2_r;
222             double m2ww1i = datam2_i;
223
224             tempr = m2ww1r - n2w1r;
225             tempi = m2ww1i - n2w1i;
226
227             datam2_r = datam1_r + tempi;
228             datam2_i = datam1_i - tempr;
229             datam1_r = datam1_r - tempi;
230             datam1_i = datam1_i + tempr;
231
232             tempr = n2w1r + m2ww1r;
233             tempi = n2w1i + m2ww1i;
234
235             datan2_r = datan1_r - tempr;
236             datan2_i = datan1_i - tempi;
237             datan1_r = datan1_r + tempr;
238             datan1_i = datan1_i + tempi;
239
240             data[m] = datam2_r;
241             data[m + 1] = datam2_i;
242             data[n] = datan2_r;
243             data[n + 1] = datan2_i;
244
245             n -= nnstep;
246             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 ww1 = w[iii++];
262     // double wwr1 = wr * wr1 - wi * wi1; // these numbers can be
263     // precomputed!!!
264     // double ww1 = 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 * ww1;
292         double m2ww1i = datam2_r * ww1 + 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];

```

```

371     n += nnstep;
372     m += nnstep;
373     double datam2_r = data[m];
374     double datam2_i = data[m + 1];
375     double datan2_r = data[n];
376     double datan2_i = data[n + 1];
377
378     double tempr = datam1_r;
379     double tempi = datam1_i;
380
381     datam1_r = datan1_r - tempr;
382     datam1_i = datan1_i - tempi;
383     datan1_r = datan1_r + tempr;
384     datan1_i = datan1_i + tempi;
385
386     double n2w1r = datan2_r;
387     double n2w1i = datan2_i;
388     double m2ww1r = datam2_r;
389     double m2ww1i = datam2_i;
390
391     tempr = n2w1r - m2ww1r;
392     tempi = n2w1i - m2ww1i;
393
394     datam2_r = datam1_r + tempi;
395     datam2_i = datam1_i - tempr;
396     datam1_r = datam1_r - tempi;
397     datam1_i = datam1_i + tempr;
398
399     tempr = n2w1r + m2ww1r;
400     tempi = n2w1i + m2ww1i;
401
402     datan2_r = datan1_r - tempr;
403     datan2_i = datan1_i - tempi;
404     datan1_r = datan1_r + tempr;
405     datan1_i = datan1_i + tempi;
406
407     data[m] = datam2_r;
408     data[m + 1] = datam2_i;
409     data[n] = datan2_r;
410     data[n + 1] = datan2_i;
411
412     n -= nnstep;
413     m -= nnstep;
414     data[m] = datam1_r;
415     data[m + 1] = datam1_i;
416     data[n] = datan1_r;
417     data[n + 1] = datan1_i;
418
419 }
420
421 }
422 for (int j = 2; j < jmax; j += 2) {
423     double wr = w[i++];
424     double wi = w[i++];
425     double wr1 = w[ii++];
426     double wi1 = w[ii++];
427     double wwr1 = w[iii++];
428     double ww1i = w[iii++];
429     // double wwr1 = wr * wr1 - wi * wi1; // these numbers can be
430     // precomputed!!!
431     // double ww1i = wr * wi1 + wi * wr1;
432

```

```

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 i += jmax << 1;
492
493 }
494

```

```

495         calcF2E(fftFrameSize, data, i, nstep, w);
496     }
497
498 // Perform Factor-4 Decomposition with 3 * complex operators and 8 +/-
499 // complex operators
500 private final static void calcF4FE(int fftFrameSize, double[] data, int i,
501     int nstep, double[] w) {
502     final int fftFrameSize2 = fftFrameSize << 1; // 2*fftFrameSize;
503     // Factor-4 Decomposition
504
505     int w_len = w.length >> 1;
506     while (nstep < fftFrameSize2) {
507
508         int jmax = nstep;
509         int nnstep = nstep << 1;
510         if (nnstep == fftFrameSize2) {
511             // Factor-4 Decomposition not possible
512             calcF2E(fftFrameSize, data, i, nstep, w);
513             return;
514         }
515         nstep <= 2;
516         int ii = i + jmax;
517         int iii = i + w_len;
518         for (int n = 0; n < jmax; n += 2) {
519             double wr = w[i++];
520             double wi = w[i++];
521             double wr1 = w[ii++];
522             double wi1 = w[ii++];
523             double wwr1 = w[iii++];
524             double ww1 = w[iii++];
525             // double wwr1 = wr * wr1 - wi * wi1; // these numbers can be
526             // precomputed!!!
527             // double ww1 = wr * wi1 + wi * wr1;
528
529             int m = n + jmax;
530
531             double datam1_r = data[m];
532             double datam1_i = data[m + 1];
533             double datan1_r = data[n];
534             double datan1_i = data[n + 1];
535
536             n += nnstep;
537             m += nnstep;
538             double datam2_r = data[m];
539             double datam2_i = data[m + 1];
540             double datan2_r = data[n];
541             double datan2_i = data[n + 1];
542
543             double tempr = datam1_r * wr - datam1_i * wi;
544             double tempi = datam1_r * wi + datam1_i * wr;
545
546             datam1_r = datan1_r - tempr;
547             datam1_i = datan1_i - tempi;
548             datan1_r = datan1_r + tempr;
549             datan1_i = datan1_i + tempi;
550
551             double n2w1r = datan2_r * wr1 - datan2_i * wi1;
552             double n2w1i = datan2_r * wi1 + datan2_i * wr1;
553             double m2ww1r = datam2_r * wwr1 - datam2_i * ww1;
554             double m2ww1i = datam2_r * ww1 + datam2_i * wwr1;

```

```

557
558     tempr = m2ww1r - n2w1r;
559     tempi = m2ww1i - n2w1i;
560
561     datam2_r = datam1_r + tempi;
562     datam2_i = datam1_i - tempr;
563     datam1_r = datam1_r - tempi;
564     datam1_i = datam1_i + tempr;
565
566     tempr = n2w1r + m2ww1r;
567     tempi = n2w1i + m2ww1i;
568
569     datan2_r = datan1_r - tempr;
570     datan2_i = datan1_i - tempi;
571     datan1_r = datan1_r + tempr;
572     datan1_i = datan1_i + tempi;
573
574     data[m] = datam2_r;
575     data[m + 1] = datam2_i;
576     data[n] = datan2_r;
577     data[n + 1] = datan2_i;
578
579     n -= nnstep;
580     m -= nnstep;
581     data[m] = datam1_r;
582     data[m + 1] = datam1_i;
583     data[n] = datan1_r;
584     data[n + 1] = datan1_i;
585
586 }
587
588 i += jmax << 1;
589
590 }
591
592 }
593
594 // Perform Factor-4 Decomposition with 3 * complex operators and 8 +/-
595 // complex operators
596 private final static void calcF4IE(int fftFrameSize, double[] data, int i,
597     int nstep, double[] w) {
598     final int fftFrameSize2 = fftFrameSize << 1; // 2*fftFrameSize;
599     // Factor-4 Decomposition
600
601     int w_len = w.length >> 1;
602     while (nstep < fftFrameSize2) {
603
604         int jmax = nstep;
605         int nnstep = nstep << 1;
606         if (nnstep == fftFrameSize2) {
607             // Factor-4 Decomposition not possible
608             calcF2E(fftFrameSize, data, i, nstep, w);
609             return;
610         }
611         nstep <= 2;
612         int ii = i + jmax;
613         int iii = i + w_len;
614         for (int n = 0; n < jmax; n += 2) {
615             double wr = w[i++];
616             double wi = w[i++];
617             double wr1 = w[ii++];
618             double wi1 = w[ii++];

```

```

619     double wwr1 = w[iii++];
620     double wwi1 = w[iii++];
621     // double wwr1 = wr * wr1 - wi * wi1; // these numbers can be
622     // precomputed!!!
623     // double wwi1 = wr * wi1 + wi * wr1;
624
625     int m = n + jmax;
626
627     double datam1_r = data[m];
628     double datam1_i = data[m + 1];
629     double datan1_r = data[n];
630     double datan1_i = data[n + 1];
631
632     n += nnstep;
633     m += nnstep;
634     double datam2_r = data[m];
635     double datam2_i = data[m + 1];
636     double datan2_r = data[n];
637     double datan2_i = data[n + 1];
638
639     double tempr = datam1_r * wr - datam1_i * wi;
640     double tempi = datam1_r * wi + datam1_i * wr;
641
642     datam1_r = datan1_r - tempr;
643     datam1_i = datan1_i - tempi;
644     datan1_r = datan1_r + tempr;
645     datan1_i = datan1_i + tempi;
646
647     double n2w1r = datan2_r * wr1 - datan2_i * wi1;
648     double n2w1i = datan2_r * wi1 + datan2_i * wr1;
649     double m2ww1r = datam2_r * wwr1 - datam2_i * wwi1;
650     double m2ww1i = datam2_r * wwi1 + datam2_i * wwr1;
651
652     tempr = n2w1r - m2ww1r;
653     tempi = n2w1i - m2ww1i;
654
655     datam2_r = datam1_r + tempi;
656     datam2_i = datam1_i - tempr;
657     datam1_r = datam1_r - tempi;
658     datam1_i = datam1_i + tempr;
659
660     tempr = n2w1r + m2ww1r;
661     tempi = n2w1i + m2ww1i;
662
663     datan2_r = datan1_r - tempr;
664     datan2_i = datan1_i - tempi;
665     datan1_r = datan1_r + tempr;
666     datan1_i = datan1_i + tempi;
667
668     data[m] = datam2_r;
669     data[m + 1] = datam2_i;
670     data[n] = datan2_r;
671     data[n + 1] = datan2_i;
672
673     n -= nnstep;
674     m -= nnstep;
675     data[m] = datam1_r;
676     data[m + 1] = datam1_i;
677     data[n] = datan1_r;
678     data[n + 1] = datan1_i;
679
680 }

```

```

681         i += jmax << 1;
682     }
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

```
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.io.IOException;
28
29 /**
30 * This exception is used when a file contains illegal or unexpected data.
31 *
32 * @author Karl Helgason
33 */
34 public class InvalidDataException extends IOException {
35
36     private static final long serialVersionUID = 1L;
37
38     public InvalidDataException() {
39         super("Invalid_Data!");
40     }
41
42     public InvalidDataException(String s) {
43         super(s);
44     }
45 }
```

34 com/sun/media/sound/InvalidFormatException.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 /**
28 * This exception is used when a reader is used to read file of a format
29 * it doesn't understand or support.
30 *
31 * @author Karl Helgason
32 */
33 public class InvalidFormatException extends InvalidDataException {
34
35     private static final long serialVersionUID = 1L;
36
37     public InvalidFormatException() {
38         super("Invalid_format!");
39     }
40
41     public InvalidFormatException(String s) {
42         super(s);
43     }
44 }
```

35 com/sun/media/sound/JARSoundbankReader.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.io.BufferedReader;
28 import java.io.File;
29 import java.io.IOException;
30 import java.io.InputStream;
31 import java.io.InputStreamReader;
32 import java.net.URL;
33 import java.net.URLClassLoader;
34 import java.util.ArrayList;
35 import javax.sound.midi.InvalidMidiDataException;
36 import javax.sound.midi.Soundbank;
37 import javax.sound.midi.spi.SoundbankReader;
38
39 /**
40 * JarSoundbankReader is used to read sounbank object from jar files.
41 *
42 * @author Karl Helgason
43 */
44 public class JARSoundbankReader extends SoundbankReader {
45
46     public boolean isZIP(URL url) {
47         boolean ok = false;
48         try {
49             InputStream stream = url.openStream();
50             try {
51                 byte[] buff = new byte[4];
52                 ok = stream.read(buff) == 4;
53                 if (ok) {
54                     ok = (buff[0] == 0x50
55                         && buff[1] == 0x4b
56                         && buff[2] == 0x03
57                         && buff[3] == 0x04);
58                 }
59             } finally {
60                 stream.close();
61             }
62         }
```

```

61     }
62     } catch (IOException e) {
63     }
64     return ok;
65 }
66
67 public Soundbank getSoundbank(URL url)
68     throws InvalidMidiDataException, IOException {
69     if (!isZIP(url))
70         return null;
71     ArrayList<Soundbank> soundbanks = new ArrayList<Soundbank>();
72     URLClassLoader ucl = URLClassLoader.newInstance(new URL[]{url});
73     InputStream stream = ucl.getResourceAsStream(
74         "META-INF/services/javax.sound.midi.Soundbank");
75     if (stream == null)
76         return null;
77     try
78     {
79         BufferedReader r = new BufferedReader(new InputStreamReader(stream));
80         String line = r.readLine();
81         while (line != null) {
82             if (!line.startsWith("#")) {
83                 try {
84                     Class c = Class.forName(line.trim(), true, ucl);
85                     Object o = c.newInstance();
86                     if (o instanceof Soundbank) {
87                         soundbanks.add((Soundbank) o);
88                     }
89                 } catch (ClassNotFoundException e) {
90                 } catch (InstantiationException e) {
91                 } catch (IllegalAccessException e) {
92                 }
93             }
94             line = r.readLine();
95         }
96     }
97     finally
98     {
99         stream.close();
100     }
101     if (soundbanks.size() == 0)
102         return null;
103     if (soundbanks.size() == 1)
104         return soundbanks.get(0);
105     SimpleSoundbank sbk = new SimpleSoundbank();
106     for (Soundbank soundbank : soundbanks)
107         sbk.addAllInstruments(soundbank);
108     return sbk;
109 }
110
111 public Soundbank getSoundbank(InputStream stream)
112     throws InvalidMidiDataException, IOException {
113     return null;
114 }
115
116 public Soundbank getSoundbank(File file)
117     throws InvalidMidiDataException, IOException {
118     return getSoundbank(file.toURI().toURL());
119 }
120 }

```

36 com/sun/media/sound/MidiDeviceReceiver.java

```
1  /*
2  * Copyright 2009 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 javax.sound.midi.MidiDevice;
28 import javax.sound.midi.Receiver;
29
30 /**
31  * A Receiver with reference to it's MidiDevice object.
32  *
33  * @author Karl Helgason
34  */
35 public interface MidiDeviceReceiver extends Receiver {
36
37     /** Obtains the MidiDevice object associated with this Receiver.
38      */
39     public MidiDevice getMidiDevice();
40
41 }
```

37 com/sun/media/sound/ModelAbstractChannelMixer.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 /**
28 * ModelAbstractChannelMixer is ready for use class to implement
29 * ModelChannelMixer interface.
30 *
31 * @author Karl Helgason
32 */
33 public abstract class ModelAbstractChannelMixer implements ModelChannelMixer {
34
35     public abstract boolean process(float[][] buffer, int offset, int len);
36
37     public abstract void stop();
38
39     public void allNotesOff() {
40     }
41
42     public void allSoundOff() {
43     }
44
45     public void controlChange(int controller, int value) {
46     }
47
48     public int getChannelPressure() {
49         return 0;
50     }
51
52     public int getController(int controller) {
53         return 0;
54     }
55
56     public boolean getMono() {
57         return false;
58     }
59
60     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

```
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.io.IOException;
28 import javax.sound.midi.Instrument;
29 import javax.sound.midi.MidiChannel;
30 import javax.sound.midi.Patch;
31 import javax.sound.midi.Soundbank;
32 import javax.sound.midi.SoundbankResource;
33 import javax.sound.midi.VoiceStatus;
34
35 /**
36  * A abstract class used to simplify creating custom ModelOscillator.
37  *
38  * @author Karl Helgason
39  */
40 public abstract class ModelAbstractOscillator
41     implements ModelOscillator, ModelOscillatorStream, Soundbank {
42
43     protected float pitch = 6000;
44     protected float samplerate;
45     protected MidiChannel channel;
46     protected VoiceStatus voice;
47     protected int noteNumber;
48     protected int velocity;
49     protected boolean on = false;
50
51     public void init() {
52     }
53
54     public void close() throws IOException {
55     }
56
57     public void noteOff(int velocity) {
58         on = false;
59     }
60
```

```

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 oscillator
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

```
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.io.ByteArrayInputStream;
28 import java.io.DataInputStream;
29 import java.io.File;
30 import java.io.IOException;
31 import java.io.InputStream;
32 import java.io.OutputStream;
33 import java.io.RandomAccessFile;
34 import java.util.Collection;
35
36 /**
37  * This class is a pointer to a binary array either in memory or on disk.
38  *
39  * @author Karl Helgason
40  */
41 public class ModelByteBuffer {
42
43     private ModelByteBuffer root = this;
44     private File file;
45     private long fileoffset;
46     private byte[] buffer;
47     private long offset;
48     private final long len;
49
50     private class RandomFileInputStream extends InputStream {
51
52         private RandomAccessFile raf;
53         private long left;
54         private long mark = 0;
55         private long markleft = 0;
56
57         public RandomFileInputStream() throws IOException {
58             raf = new RandomAccessFile(root.file, "r");
59             raf.seek(root.fileoffset + arrayOffset());
60             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

```

```

123     public int read() throws IOException {
124         if (left == 0)
125             return -1;
126         int b = raf.read();
127         if (b == -1)
128             return -1;
129         left--;
130         return b;
131     }
132
133     public void close() throws IOException {
134         raf.close();
135     }
136 }
137
138 private ModelByteBuffer(ModelByteBuffer parent,
139     long beginIndex, long endIndex, boolean independent) {
140     this.root = parent.root;
141     this.offset = 0;
142     long parent_len = parent.len;
143     if (beginIndex < 0)
144         beginIndex = 0;
145     if (beginIndex > parent_len)
146         beginIndex = parent_len;
147     if (endIndex < 0)
148         endIndex = 0;
149     if (endIndex > parent_len)
150         endIndex = parent_len;
151     if (beginIndex > endIndex)
152         beginIndex = endIndex;
153     offset = beginIndex;
154     len = endIndex - beginIndex;
155     if (independent) {
156         buffer = root.buffer;
157         if (root.file != null) {
158             file = root.file;
159             fileoffset = root.fileoffset + arrayOffset();
160             offset = 0;
161         } else
162             offset = arrayOffset();
163         root = this;
164     }
165 }
166
167 public ModelByteBuffer(byte[] buffer) {
168     this.buffer = buffer;
169     this.offset = 0;
170     this.len = buffer.length;
171 }
172
173 public ModelByteBuffer(byte[] buffer, int offset, int len) {
174     this.buffer = buffer;
175     this.offset = offset;
176     this.len = len;
177 }
178
179 public ModelByteBuffer(File file) {
180     this.file = file;
181     this.fileoffset = 0;
182     this.len = file.length();
183 }
184

```



```

185 public ModelByteBuffer(File file, long offset, long len) {
186     this.file = file;
187     this.fileoffset = offset;
188     this.len = len;
189 }
190
191 public void writeTo(OutputStream out) throws IOException {
192     if (root.file != null && root.buffer == null) {
193         InputStream is = getInputStream();
194         byte[] buff = new byte[1024];
195         int ret;
196         while ((ret = is.read(buff)) != -1)
197             out.write(buff, 0, ret);
198     } else
199         out.write(array(), (int) arrayOffset(), (int) capacity());
200 }
201
202 public InputStream getInputStream() {
203     if (root.file != null && root.buffer == null) {
204         try {
205             return new RandomFileInputStream();
206         } catch (IOException e) {
207             //e.printStackTrace();
208             return null;
209         }
210     }
211     return new ByteArrayInputStream(array(),
212                                     (int)arrayOffset(), (int)capacity());
213 }
214
215 public ModelByteBuffer subbuffer(long beginIndex) {
216     return subbuffer(beginIndex, capacity());
217 }
218
219 public ModelByteBuffer subbuffer(long beginIndex, long endIndex) {
220     return subbuffer(beginIndex, endIndex, false);
221 }
222
223 public ModelByteBuffer subbuffer(long beginIndex, long endIndex,
224     boolean independent) {
225     return new ModelByteBuffer(this, beginIndex, endIndex, independent);
226 }
227
228 public byte[] array() {
229     return root.buffer;
230 }
231
232 public long arrayOffset() {
233     if (root != this)
234         return root.arrayOffset() + offset;
235     return offset;
236 }
237
238 public long capacity() {
239     return len;
240 }
241
242 public ModelByteBuffer getRoot() {
243     return root;
244 }
245
246 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 selffile = 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 (selffile == null || !selffile.equals(mbuff.file)) {
266                 if (raf != null) {
267                     raf.close();
268                     raf = null;
269                 }
270                 selffile = 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             mbuff.buffer = buffer;
289             mbuff.offset = 0;
290         }
291     } finally {
292         if (raf != null)
293             raf.close();
294     }
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

```
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.io.IOException;
28 import java.io.InputStream;
29 import javax.sound.sampled.AudioFormat;
30 import javax.sound.sampled.AudioInputStream;
31 import javax.sound.sampled.AudioSystem;
32 import javax.sound.sampled.AudioFormat.Encoding;
33
34 /**
35  * Wavetable oscillator for pre-loaded data.
36  *
37  * @author Karl Helgason
38  */
39 public class ModelByteBufferWavetable implements ModelWavetable {
40
41     private class Buffer8PlusInputStream extends InputStream {
42
43         private boolean bigendian;
44         private int framesize_pc;
45         int pos = 0;
46         int pos2 = 0;
47         int markpos = 0;
48         int markpos2 = 0;
49
50         public Buffer8PlusInputStream() {
51             framesize_pc = format.getFrameSize() / format.getChannels();
52             bigendian = format.isBigEndian();
53         }
54
55         public int read(byte[] b, int off, int len) throws IOException {
56             int avail = available();
57             if (avail <= 0)
58                 return -1;
59             if (len > avail)
60                 len = avail;
```

```

61     byte[] buff1 = buffer.array();
62     byte[] buff2 = buffer8.array();
63     pos += buffer.arrayOffset();
64     pos2 += buffer8.arrayOffset();
65     if (bigendian) {
66         for (int i = 0; i < len; i += (framesize_pc + 1)) {
67             System.arraycopy(buff1, pos, b, i, framesize_pc);
68             System.arraycopy(buff2, pos2, b, i + framesize_pc, 1);
69             pos += framesize_pc;
70             pos2 += 1;
71         }
72     } else {
73         for (int i = 0; i < len; i += (framesize_pc + 1)) {
74             System.arraycopy(buff2, pos2, b, i, 1);
75             System.arraycopy(buff1, pos, b, i + 1, framesize_pc);
76             pos += framesize_pc;
77             pos2 += 1;
78         }
79     }
80     pos -= buffer.arrayOffset();
81     pos2 -= buffer8.arrayOffset();
82     return len;
83 }
84
85 public long skip(long n) throws IOException {
86     int avail = available();
87     if (avail <= 0)
88         return -1;
89     if (n > avail)
90         n = avail;
91     pos += (n / (framesize_pc + 1)) * (framesize_pc);
92     pos2 += n / (framesize_pc + 1);
93     return super.skip(n);
94 }
95
96 public int read(byte[] b) throws IOException {
97     return read(b, 0, b.length);
98 }
99
100 public int read() throws IOException {
101     byte[] b = new byte[1];
102     int ret = read(b, 0, 1);
103     if (ret == -1)
104         return -1;
105     return 0 & 0xFF;
106 }
107
108 public boolean markSupported() {
109     return true;
110 }
111
112 public int available() throws IOException {
113     return (int)buffer.capacity() + (int)buffer8.capacity() - pos - pos2;
114 }
115
116 public synchronized void mark(int readlimit) {
117     markpos = pos;
118     markpos2 = pos2;
119 }
120
121 public synchronized void reset() throws IOException {
122     pos = markpos;

```

```

123         pos2 = markpos2;
124
125     }
126 }
127
128 private float loopStart = -1;
129 private float loopLength = -1;
130 private ModelByteBuffer buffer;
131 private ModelByteBuffer buffer8 = null;
132 private AudioFormat format = null;
133 private float pitchcorrection = 0;
134 private float attenuation = 0;
135 private int loopType = LOOP_TYPE_OFF;
136
137 public ModelByteBufferWavetable(ModelByteBuffer buffer) {
138     this.buffer = buffer;
139 }
140
141 public ModelByteBufferWavetable(ModelByteBuffer buffer,
142     float pitchcorrection) {
143     this.buffer = buffer;
144     this.pitchcorrection = pitchcorrection;
145 }
146
147 public ModelByteBufferWavetable(ModelByteBuffer buffer, AudioFormat format) {
148     this.format = format;
149     this.buffer = buffer;
150 }
151
152 public ModelByteBufferWavetable(ModelByteBuffer buffer, AudioFormat format,
153     float pitchcorrection) {
154     this.format = format;
155     this.buffer = buffer;
156     this.pitchcorrection = pitchcorrection;
157 }
158
159 public void set8BitExtensionBuffer(ModelByteBuffer buffer) {
160     buffer8 = buffer;
161 }
162
163 public ModelByteBuffer get8BitExtensionBuffer() {
164     return buffer8;
165 }
166
167 public ModelByteBuffer getBuffer() {
168     return buffer;
169 }
170
171 public AudioFormat getFormat() {
172     if (format == null) {
173         if (buffer == null)
174             return null;
175         InputStream is = buffer.getInputStream();
176         AudioFormat format = null;
177         try {
178             format = AudioSystem.getAudioFileFormat(is).getFormat();
179         } catch (Exception e) {
180             //e.printStackTrace();
181         }
182         try {
183             is.close();
184         } catch (IOException e) {

```

```

185         //e.printStackTrace();
186     }
187     return format;
188 }
189 return format;
190 }
191
192 public AudioFloatInputStream openStream() {
193     if (buffer == null)
194         return null;
195     if (format == null) {
196         InputStream is = buffer.getInputStream();
197         AudioInputStream ais = null;
198         try {
199             ais = AudioSystem.getAudioInputStream(is);
200         } catch (Exception e) {
201             //e.printStackTrace();
202             return null;
203         }
204         return AudioFloatInputStream.getInputStream(ais);
205     }
206     if (buffer.array() == null) {
207         return AudioFloatInputStream.getInputStream(new AudioInputStream(
208             buffer.getInputStream(), format,
209             buffer.capacity() / format.getFrameSize()));
210     }
211     if (buffer8 != null) {
212         if (format.getEncoding().equals(Encoding.PCM_SIGNED)
213             || format.getEncoding().equals(Encoding.PCM_UNSIGNED)) {
214             InputStream is = new Buffer8PlusInputStream();
215             AudioFormat format2 = new AudioFormat(
216                 format.getEncoding(),
217                 format.getSampleRate(),
218                 format.getSampleSizeInBits() + 8,
219                 format.getChannels(),
220                 format.getFrameSize() + (1 * format.getChannels()),
221                 format.getFrameRate(),
222                 format.isBigEndian());
223
224             AudioInputStream ais = new AudioInputStream(is, format2,
225                 buffer.capacity() / format.getFrameSize());
226             return AudioFloatInputStream.getInputStream(ais);
227         }
228     }
229     return AudioFloatInputStream.getInputStream(format, buffer.array(),
230         (int)buffer.arrayOffset(), (int)buffer.capacity());
231 }
232
233 public int getChannels() {
234     return getFormat().getChannels();
235 }
236
237 public ModelOscillatorStream open(float samplerate) {
238     // ModelWavetableOscillator doesn't support ModelOscillatorStream
239     return null;
240 }
241
242 // attenuation is in cB
243 public float getAttenuation() {
244     return attenuation;
245 }
246 // attenuation is in cB

```

```

247     public void setAttenuation(float attenuation) {
248         this.attenuation = attenuation;
249     }
250
251     public float getLoopLength() {
252         return loopLength;
253     }
254
255     public void setLoopLength(float loopLength) {
256         this.loopLength = loopLength;
257     }
258
259     public float getLoopStart() {
260         return loopStart;
261     }
262
263     public void setLoopStart(float loopStart) {
264         this.loopStart = loopStart;
265     }
266
267     public void setLoopType(int loopType) {
268         this.loopType = loopType;
269     }
270
271     public int getLoopType() {
272         return loopType;
273     }
274
275     public float getPitchcorrection() {
276         return pitchcorrection;
277     }
278
279     public void setPitchcorrection(float pitchcorrection) {
280         this.pitchcorrection = pitchcorrection;
281     }
282 }

```


41 com/sun/media/sound/ModelChannelMixer.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 javax.sound.midi.MidiChannel;
28
29 /**
30 * ModelChannelMixer is used to process channel voice mix output before going
31 * to master output.<br>
32 * It can be used to:<br>
33 * <ul>
34 * <li>Implement non-voice oriented instruments.</li>
35 * <li>Add insert effect to instruments; for example distortion effect.</li>
36 * </ul>
37 * <p>
38 * <b>Warning! Classes that implements ModelChannelMixer must be thread-safe.</b>
39 *
40 * @author Karl Helgason
41 */
42 public interface ModelChannelMixer extends MidiChannel {
43
44     // Used to process input audio from voices mix.
45     public boolean process(float[][] buffer, int offset, int len);
46
47     // Is used to trigger that this mixer is not be used
48     // and it should fade out.
49     public void stop();
50 }
```

42 com/sun/media/sound/ModelConnectionBlock.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 /**
28 * Connection blocks are used to connect source variable
29 * to a destination variable.
30 * For example Note On velocity can be connected to output gain.
31 * In DLS this is called articulator and in SoundFonts (SF2) a modulator.
32 *
33 * @author Karl Helgason
34 */
35 public class ModelConnectionBlock {
36
37     //
38     //  source1 * source2 * scale -> destination
39     //
40     private final static ModelSource[] no_sources = new ModelSource[0];
41     private ModelSource[] sources = no_sources;
42     private double scale = 1;
43     private ModelDestination destination;
44
45     public ModelConnectionBlock() {
46     }
47
48     public ModelConnectionBlock(double scale, ModelDestination destination) {
49         this.scale = scale;
50         this.destination = destination;
51     }
52
53     public ModelConnectionBlock(ModelSource source,
54                                 ModelDestination destination) {
55         if (source != null) {
56             this.sources = new ModelSource[1];
57             this.sources[0] = source;
58         }
59         this.destination = destination;
60     }
61 }
```

```

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

```
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 /**
28 * This class is used to identify destinations in connection blocks,
29 * see ModelConnectionBlock.
30 *
31 * @author Karl Helgason
32 */
33 public class ModelDestination {
34
35     public static final ModelIdentifier DESTINATION_NONE = null;
36     public static final ModelIdentifier DESTINATION_KEYNUMBER
37         = new ModelIdentifier("noteon", "keynumber");
38     public static final ModelIdentifier DESTINATION_VELOCITY
39         = new ModelIdentifier("noteon", "velocity");
40     public static final ModelIdentifier DESTINATION_PITCH
41         = new ModelIdentifier("osc", "pitch"); // cent
42     public static final ModelIdentifier DESTINATION_GAIN
43         = new ModelIdentifier("mixer", "gain"); // cB
44     public static final ModelIdentifier DESTINATION_PAN
45         = new ModelIdentifier("mixer", "pan"); // 0.1 %
46     public static final ModelIdentifier DESTINATION_REVERB
47         = new ModelIdentifier("mixer", "reverb"); // 0.1 %
48     public static final ModelIdentifier DESTINATION_CHORUS
49         = new ModelIdentifier("mixer", "chorus"); // 0.1 %
50     public static final ModelIdentifier DESTINATION_LF01_DELAY
51         = new ModelIdentifier("lfo", "delay", 0); // timecent
52     public static final ModelIdentifier DESTINATION_LF01_FREQ
53         = new ModelIdentifier("lfo", "freq", 0); // cent
54     public static final ModelIdentifier DESTINATION_LF02_DELAY
55         = new ModelIdentifier("lfo", "delay", 1); // timecent
56     public static final ModelIdentifier DESTINATION_LF02_FREQ
57         = new ModelIdentifier("lfo", "freq", 1); // cent
58     public static final ModelIdentifier DESTINATION_EG1_DELAY
59         = new ModelIdentifier("eg", "delay", 0); // timecent
60     public static final ModelIdentifier DESTINATION_EG1_ATTACK
```

```

61         = new ModelIdentifier("eg", "attack", 0); // timecent
62 public static final ModelIdentifier DESTINATION_EG1_HOLD
63         = new ModelIdentifier("eg", "hold", 0); // timecent
64 public static final ModelIdentifier DESTINATION_EG1_DECAY
65         = new ModelIdentifier("eg", "decay", 0); // timecent
66 public static final ModelIdentifier DESTINATION_EG1_SUSTAIN
67         = new ModelIdentifier("eg", "sustain", 0);
68         // 0.1 % (I want this to be value not %)
69 public static final ModelIdentifier DESTINATION_EG1_RELEASE
70         = new ModelIdentifier("eg", "release", 0); // timecent
71 public static final ModelIdentifier DESTINATION_EG1_SHUTDOWN
72         = new ModelIdentifier("eg", "shutdown", 0); // timecent
73 public static final ModelIdentifier DESTINATION_EG2_DELAY
74         = new ModelIdentifier("eg", "delay", 1); // timecent
75 public static final ModelIdentifier DESTINATION_EG2_ATTACK
76         = new ModelIdentifier("eg", "attack", 1); // timecent
77 public static final ModelIdentifier DESTINATION_EG2_HOLD
78         = new ModelIdentifier("eg", "hold", 1); // 0.1 %
79 public static final ModelIdentifier DESTINATION_EG2_DECAY
80         = new ModelIdentifier("eg", "decay", 1); // timecent
81 public static final ModelIdentifier DESTINATION_EG2_SUSTAIN
82         = new ModelIdentifier("eg", "sustain", 1);
83         // 0.1 % ( I want this to be value not %)
84 public static final ModelIdentifier DESTINATION_EG2_RELEASE
85         = new ModelIdentifier("eg", "release", 1); // timecent
86 public static final ModelIdentifier DESTINATION_EG2_SHUTDOWN
87         = new ModelIdentifier("eg", "shutdown", 1); // timecent
88 public static final ModelIdentifier DESTINATION_FILTER_FREQ
89         = new ModelIdentifier("filter", "freq", 0); // cent
90 public static final ModelIdentifier DESTINATION_FILTER_Q
91         = new ModelIdentifier("filter", "q", 0); // cB
92 private ModelIdentifier destination = DESTINATION_NONE;
93 private ModelTransform transform = new ModelStandardTransform();
94
95 public ModelDestination() {
96 }
97
98 public ModelDestination(ModelIdentifier id) {
99     destination = id;
100 }
101
102 public ModelIdentifier getIdentifier() {
103     return destination;
104 }
105
106 public void setIdentifier(ModelIdentifier destination) {
107     this.destination = destination;
108 }
109
110 public ModelTransform getTransform() {
111     return transform;
112 }
113
114 public void setTransform(ModelTransform transform) {
115     this.transform = transform;
116 }
117 }

```

44 com/sun/media/sound/ModelDirectedPlayer.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 /**
28 * ModelDirectedPlayer is the one who is directed by ModelDirector
29 * to play ModelPerformer objects.
30 *
31 * @author Karl Helgason
32 */
33 public interface ModelDirectedPlayer {
34
35     public void play(int performerIndex, ModelConnectionBlock[] connectionBlocks);
36 }
```

45 com/sun/media/sound/ModelDirector.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 /**
28 * A director chooses what performers should be played for each note on
29 * and note off events.
30 *
31 * ModelInstrument can implement custom performer who chooses what performers
32 * to play for example by sustain pedal is off or on.
33 *
34 * The default director (ModelStandardDirector) chooses performers
35 * by there keyfrom,keyto,velfrom,velto properties.
36 *
37 * @author Karl Helgason
38 */
39 public interface ModelDirector {
40
41     public void noteOn(int noteNumber, int velocity);
42
43     public void noteOff(int noteNumber, int velocity);
44
45     public void close();
46 }
```


46 com/sun/media/sound/ModelIdentifier.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 /**
28 * This class stores the identity of source and destinations in connection
29 * blocks, see ModelConnectionBlock.
30 *
31 * @author Karl Helgason
32 */
33 public class ModelIdentifier {
34
35     /*
36      * Object      Variable
37      * -----
38      *
39      * // INPUT parameters
40      * noteon      keynumber          7 bit midi value
41      *            velocity            7 bit midi vale
42      *            on                  1 or 0
43      *
44      * midi        pitch              14 bit midi value
45      *            channel_pressure    7 bit midi value
46      *            poly_pressure       7 bit midi value
47      *
48      * midi_cc     0 (midi control #0  7 bit midi value
49      *            1 (midi control #1    7 bit midi value
50      *            ...
51      *            127 (midi control #127 7 bit midi value
52      *
53      * midi_rpn    0 (midi rpn control #0) 14 bit midi value
54      *            1 (midi rpn control #1) 14 bit midi value
55      *            ....
56      *
57      * // DAHDSR envelope generator
58      * eg          (null)
59      *            delay              timecent
60      *            attack             timecent
```

```

61      *          hold          timecent
62      *          decay         timecent
63      *          sustain       0.1 %
64      *          release       timecent
65      *
66      * // Low frequency oscillirator (sine wave)
67      * lfo          (null)
68      *          delay         timcent
69      *          freq          cent
70      *
71      * // Resonance LowPass Filter 6dB slope
72      * filter      (null) (output/input)
73      *          freq          cent
74      *          q            cB
75      *
76      * // The oscillator with preloaded wavetable data
77      * osc          (null)
78      *          pitch         cent
79      *
80      * // Output mixer pins
81      * mixer      gain          cB
82      *          pan          0.1 %
83      *          reverb       0.1 %
84      *          chorus       0.1 %
85      *
86      */
87      private String object = null;
88      private String variable = null;
89      private int instance = 0;
90
91      public ModelIdentifier(String object) {
92          this.object = object;
93      }
94
95      public ModelIdentifier(String object, int instance) {
96          this.object = object;
97          this.instance = instance;
98      }
99
100     public ModelIdentifier(String object, String variable) {
101         this.object = object;
102         this.variable = variable;
103     }
104
105
106     public ModelIdentifier(String object, String variable, int instance) {
107         this.object = object;
108         this.variable = variable;
109         this.instance = instance;
110     }
111
112
113     public int getInstance() {
114         return instance;
115     }
116
117     public void setInstance(int instance) {
118         this.instance = instance;
119     }
120
121     public String getObject() {
122         return object;

```

```

123     }
124
125     public void setObject(String object) {
126         this.object = object;
127     }
128
129     public String getVariable() {
130         return variable;
131     }
132
133     public void setVariable(String variable) {
134         this.variable = variable;
135     }
136
137     public int hashCode() {
138         int hashCode = instance;
139         if(object != null) hashCode |= object.hashCode();
140         if(variable != null) hashCode |= variable.hashCode();
141         return hashCode;
142     }
143
144     public boolean equals(Object obj) {
145         if (!(obj instanceof ModelIdentifier))
146             return false;
147
148         ModelIdentifier mobj = (ModelIdentifier)obj;
149         if ((object == null) != (mobj.object == null))
150             return false;
151         if ((variable == null) != (mobj.variable == null))
152             return false;
153         if (mobj.getInstance() != getInstance())
154             return false;
155         if (!(object == null || object.equals(mobj.object)))
156             return false;
157         if (!(variable == null || variable.equals(mobj.variable)))
158             return false;
159         return true;
160     }
161
162     public String toString() {
163         if (variable == null) {
164             return object + "[" + instance + "]";
165         } else {
166             return object + "[" + instance + "]" + "." + variable;
167         }
168     }
169 }

```

47 com/sun/media/sound/ModelInstrument.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 javax.sound.midi.Instrument;
28 import javax.sound.midi.MidiChannel;
29 import javax.sound.midi.Patch;
30 import javax.sound.midi.Soundbank;
31 import javax.sound.sampled.AudioFormat;
32
33 /**
34  * The model instrument class.
35  *
36  * <p>The main methods to override are:<br>
37  * getPerformer, getDirector, getChannelMixer.
38  *
39  * <p>Performers are used to define what voices which will
40  * playback when using the instrument.<br>
41  *
42  * ChannelMixer is used to add channel-wide processing
43  * on voices output or to define non-voice oriented instruments.<br>
44  *
45  * Director is used to change how the synthesizer
46  * chooses what performers to play on midi events.
47  *
48  * @author Karl Helgason
49  */
50 public abstract class ModelInstrument extends Instrument {
51
52     protected ModelInstrument(Soundbank soundbank, Patch patch, String name,
53                               Class<?> dataClass) {
54         super(soundbank, patch, name, dataClass);
55     }
56
57     public ModelDirector getDirector(ModelPerformer[] performers,
58                                     MidiChannel channel, ModelDirectedPlayer player) {
59         return new ModelStandardIndexedDirector(performers, player);
60     }
61 }
```

```

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();

```

```

123     if (bank >> 7 == 0x78 || bank >> 7 == 0x79) {
124         boolean[] ch = new boolean[16];
125         for (int i = 0; i < ch.length; i++)
126             ch[i] = true;
127         return ch;
128     }
129
130     boolean[] ch = new boolean[16];
131     for (int i = 0; i < ch.length; i++)
132         ch[i] = true;
133     ch[9] = false;
134     return ch;
135 }
136 }

```

48 com/sun/media/sound/ModelInstrumentComparator.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.Comparator;
28 import javax.sound.midi.Instrument;
29 import javax.sound.midi.Patch;
30
31 /**
32  * Instrument comparator class.
33  * Used to order instrument by program, bank, percussion.
34  *
35  * @author Karl Helgason
36  */
37 public class ModelInstrumentComparator implements Comparator<Instrument> {
38
39     public int compare(Instrument arg0, Instrument arg1) {
40         Patch p0 = arg0.getPatch();
41         Patch p1 = arg1.getPatch();
42         int a = p0.getBank() * 128 + p0.getProgram();
43         int b = p1.getBank() * 128 + p1.getProgram();
44         if (p0 instanceof ModelPatch) {
45             a += ((ModelPatch)p0).isPercussion() ? 2097152 : 0;
46         }
47         if (p1 instanceof ModelPatch) {
48             b += ((ModelPatch)p1).isPercussion() ? 2097152 : 0;
49         }
50         return a - b;
51     }
52 }
```

49 com/sun/media/sound/ModelMappedInstrument.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 javax.sound.midi.MidiChannel;
28 import javax.sound.midi.Patch;
29 import javax.sound.sampled.AudioFormat;
30
31 /**
32  * This class is used to map instrument to another patch.
33  *
34  * @author Karl Helgason
35  */
36 public class ModelMappedInstrument extends ModelInstrument {
37
38     private ModelInstrument ins;
39
40     public ModelMappedInstrument(ModelInstrument ins, Patch patch) {
41         super(ins.getSoundbank(), patch, ins.getName(), ins.getDataClass());
42         this.ins = ins;
43     }
44
45     public Object getData() {
46         return ins.getData();
47     }
48
49     public ModelPerformer[] getPerformers() {
50         return ins.getPerformers();
51     }
52
53     public ModelDirector getDirector(ModelPerformer[] performers,
54         MidiChannel channel, ModelDirectedPlayer player) {
55         return ins.getDirector(performers, channel, player);
56     }
57
58     public ModelChannelMixer getChannelMixer(MidiChannel channel,
59         AudioFormat format) {
60         return ins.getChannelMixer(channel, format);
61     }
62 }
```


61 }

62 }

50 com/sun/media/sound/ModelOscillator.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 /**
28 * This interface is used for oscillators.
29 * See example in ModelDefaultOscillator which is a wavetable oscillator.
30 *
31 * @author Karl Helgason
32 */
33 public interface ModelOscillator {
34
35     public int getChannels();
36
37     /**
38      * Attenuation is in cB.
39      * @return
40      */
41     public float getAttenuation();
42
43     public ModelOscillatorStream open(float samplerate);
44 }
```

51 com/sun/media/sound/ModelOscillatorStream.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.io.IOException;
28 import javax.sound.midi.MidiChannel;
29 import javax.sound.midi.VoiceStatus;
30
31 /**
32 * This interface is used for audio streams from ModelOscillator.
33 *
34 * @author Karl Helgason
35 */
36 public interface ModelOscillatorStream {
37
38     public void setPitch(float pitch); // Pitch is in cents!
39
40     public void noteOn(MidiChannel channel, VoiceStatus voice, int noteNumber,
41                       int velocity);
42
43     public void noteOff(int velocity);
44
45     public int read(float[][] buffer, int offset, int len) throws IOException;
46
47     public void close() throws IOException;
48 }
```

52 com/sun/media/sound/ModelPatch.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 javax.sound.midi.Patch;
28
29 /**
30 * A extended patch object that has isPercussion function.
31 * Which is necessary to identify percussion instruments
32 * from melodic instruments.
33 *
34 * @author Karl Helgason
35 */
36 public class ModelPatch extends Patch {
37
38     private boolean percussion = false;
39
40     public ModelPatch(int bank, int program) {
41         super(bank, program);
42     }
43
44     public ModelPatch(int bank, int program, boolean percussion) {
45         super(bank, program);
46         this.percussion = percussion;
47     }
48
49     public boolean isPercussion() {
50         return percussion;
51     }
52 }
```

53 com/sun/media/sound/ModelPerformer.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.ArrayList;
28 import java.util.List;
29
30 /**
31 * This class is used to define how to synthesize audio in universal maner
32 * for both SF2 and DLS instruments.
33 *
34 * @author Karl Helgason
35 */
36 public class ModelPerformer {
37
38     private List<ModelOscillator> oscillators = new ArrayList<ModelOscillator>();
39     private List<ModelConnectionBlock> connectionBlocks
40         = new ArrayList<ModelConnectionBlock>();
41     private int keyFrom = 0;
42     private int keyTo = 127;
43     private int velFrom = 0;
44     private int velTo = 127;
45     private int exclusiveClass = 0;
46     private boolean releaseTrigger = false;
47     private boolean selfNonExclusive = false;
48     private Object userObject = null;
49     private boolean addDefaultConnections = true;
50     private String name = null;
51
52     public String getName() {
53         return name;
54     }
55
56     public void setName(String name) {
57         this.name = name;
58     }
59
60     public List<ModelConnectionBlock> getConnectionBlocks() {
```

```

61         return connectionBlocks;
62     }
63
64     public void setConnectionBlocks(List<ModelConnectionBlock> connectionBlocks) {
65         this.connectionBlocks = connectionBlocks;
66     }
67
68     public List<ModelOscillator> getOscillators() {
69         return oscillators;
70     }
71
72     public int getExclusiveClass() {
73         return exclusiveClass;
74     }
75
76     public void setExclusiveClass(int exclusiveClass) {
77         this.exclusiveClass = exclusiveClass;
78     }
79
80     public boolean isSelfNonExclusive() {
81         return selfNonExclusive;
82     }
83
84     public void setSelfNonExclusive(boolean selfNonExclusive) {
85         this.selfNonExclusive = selfNonExclusive;
86     }
87
88     public int getKeyFrom() {
89         return keyFrom;
90     }
91
92     public void setKeyFrom(int keyFrom) {
93         this.keyFrom = keyFrom;
94     }
95
96     public int getKeyTo() {
97         return keyTo;
98     }
99
100    public void setKeyTo(int keyTo) {
101        this.keyTo = keyTo;
102    }
103
104    public int getVelFrom() {
105        return velFrom;
106    }
107
108    public void setVelFrom(int velFrom) {
109        this.velFrom = velFrom;
110    }
111
112    public int getVelTo() {
113        return velTo;
114    }
115
116    public void setVelTo(int velTo) {
117        this.velTo = velTo;
118    }
119
120    public boolean isReleaseTriggered() {
121        return releaseTrigger;
122    }

```

```
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

```
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 /**
28 * This class is used to identify sources in connection blocks,
29 * see ModelConnectionBlock.
30 *
31 * @author Karl Helgason
32 */
33 public class ModelSource {
34
35     public static final ModelIdentifier SOURCE_NONE = null;
36     public static final ModelIdentifier SOURCE_NOTEON_KEYNUMBER =
37         new ModelIdentifier("noteon", "keynumber"); // midi keynumber
38     public static final ModelIdentifier SOURCE_NOTEON_VELOCITY =
39         new ModelIdentifier("noteon", "velocity"); // midi velocity
40     public static final ModelIdentifier SOURCE_EG1 =
41         new ModelIdentifier("eg", null, 0);
42     public static final ModelIdentifier SOURCE_EG2 =
43         new ModelIdentifier("eg", null, 1);
44     public static final ModelIdentifier SOURCE_LF01 =
45         new ModelIdentifier("lfo", null, 0);
46     public static final ModelIdentifier SOURCE_LF02 =
47         new ModelIdentifier("lfo", null, 1);
48     public static final ModelIdentifier SOURCE_MIDI_PITCH =
49         new ModelIdentifier("midi", "pitch", 0); // (0..16383)
50     public static final ModelIdentifier SOURCE_MIDI_CHANNEL_PRESSURE =
51         new ModelIdentifier("midi", "channel_pressure", 0); // (0..127)
52 //     public static final ModelIdentifier SOURCE_MIDI_MONO_PRESSURE =
53 //         new ModelIdentifier("midi", "mono_pressure", 0); // (0..127)
54     public static final ModelIdentifier SOURCE_MIDI_POLY_PRESSURE =
55         new ModelIdentifier("midi", "poly_pressure", 0); // (0..127)
56     public static final ModelIdentifier SOURCE_MIDI_CC_0 =
57         new ModelIdentifier("midi_cc", "0", 0); // (0..127)
58     public static final ModelIdentifier SOURCE_MIDI_RPN_0 =
59         new ModelIdentifier("midi_rpn", "0", 0); // (0..16383)
60     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

```
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 /**
28 * A standard director who chooses performers
29 * by there keyfrom,keyto,velfrom,velto properties.
30 *
31 * @author Karl Helgason
32 */
33 public class ModelStandardDirector implements ModelDirector {
34
35     ModelPerformer[] performers;
36     ModelDirectedPlayer player;
37     boolean noteOnUsed = false;
38     boolean noteOffUsed = false;
39
40     public ModelStandardDirector(ModelPerformer[] performers,
41                                 ModelDirectedPlayer player) {
42         this.performers = performers;
43         this.player = player;
44         for (int i = 0; i < performers.length; i++) {
45             ModelPerformer p = performers[i];
46             if (p.isReleaseTriggered()) {
47                 noteOffUsed = true;
48             } else {
49                 noteOnUsed = true;
50             }
51         }
52     }
53
54     public void close() {
55     }
56
57     public void noteOff(int noteNumber, int velocity) {
58         if (!noteOffUsed)
59             return;
60         for (int i = 0; i < performers.length; i++) {
```

```

61         ModelPerformer p = performers[i];
62         if (p.getKeyFrom() <= noteNumber && p.getKeyTo() >= noteNumber) {
63             if (p.getVelFrom() <= velocity && p.getVelTo() >= velocity) {
64                 if (p.isReleaseTriggered()) {
65                     player.play(i, null);
66                 }
67             }
68         }
69     }
70 }
71
72 public void noteOn(int noteNumber, int velocity) {
73     if (!noteOnUsed)
74         return;
75     for (int i = 0; i < performers.length; i++) {
76         ModelPerformer p = performers[i];
77         if (p.getKeyFrom() <= noteNumber && p.getKeyTo() >= noteNumber) {
78             if (p.getVelFrom() <= velocity && p.getVelTo() >= velocity) {
79                 if (!p.isReleaseTriggered()) {
80                     player.play(i, null);
81                 }
82             }
83         }
84     }
85 }
86 }

```

56 com/sun/media/sound/ModelStandardIndexedDirector.java

```
1  /*
2  * Copyright 2010 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 /**
28 * A standard indexed director who chooses performers
29 * by there keyfrom,keyto,velfrom,velto properties.
30 *
31 * @author Karl Helgason
32 */
33 public class ModelStandardIndexedDirector implements ModelDirector {
34
35     ModelPerformer[] performers;
36     ModelDirectedPlayer player;
37     boolean noteOnUsed = false;
38     boolean noteOffUsed = false;
39
40     // Variables needed for index
41     byte[][] trantables;
42     int[] counters;
43     int[][] mat;
44
45     public ModelStandardIndexedDirector(ModelPerformer[] performers,
46         ModelDirectedPlayer player) {
47         this.performers = performers;
48         this.player = player;
49         for (int i = 0; i < performers.length; i++) {
50             ModelPerformer p = performers[i];
51             if (p.isReleaseTriggered()) {
52                 noteOffUsed = true;
53             } else {
54                 noteOnUsed = true;
55             }
56         }
57         buildindex();
58     }
59
60     private int[] lookupIndex(int x, int y) {
```

```

61     if ((x >= 0) && (x < 128) && (y >= 0) && (y < 128)) {
62         int xt = trantables[0][x];
63         int yt = trantables[1][y];
64         if (xt != -1 && yt != -1) {
65             return mat[xt + yt * counters[0]];
66         }
67     }
68     return null;
69 }
70
71 private int restrict(int value) {
72     if(value < 0) return 0;
73     if(value > 127) return 127;
74     return value;
75 }
76
77 private void buildindex() {
78     trantables = new byte[2][129];
79     counters = new int[trantables.length];
80     for (ModelPerformer performer : performers) {
81         int keyFrom = performer.getKeyFrom();
82         int keyTo = performer.getKeyTo();
83         int velFrom = performer.getVelFrom();
84         int velTo = performer.getVelTo();
85         if (keyFrom > keyTo) continue;
86         if (velFrom > velTo) continue;
87         keyFrom = restrict(keyFrom);
88         keyTo = restrict(keyTo);
89         velFrom = restrict(velFrom);
90         velTo = restrict(velTo);
91         trantables[0][keyFrom] = 1;
92         trantables[0][keyTo + 1] = 1;
93         trantables[1][velFrom] = 1;
94         trantables[1][velTo + 1] = 1;
95     }
96     for (int d = 0; d < trantables.length; d++) {
97         byte[] trantable = trantables[d];
98         int transize = trantable.length;
99         for (int i = transize - 1; i >= 0; i--) {
100             if (trantable[i] == 1) {
101                 trantable[i] = -1;
102                 break;
103             }
104             trantable[i] = -1;
105         }
106         int counter = -1;
107         for (int i = 0; i < transize; i++) {
108             if (trantable[i] != 0) {
109                 counter++;
110                 if (trantable[i] == -1)
111                     break;
112             }
113             trantable[i] = (byte) counter;
114         }
115         counters[d] = counter;
116     }
117     mat = new int[counters[0] * counters[1]][];
118     int ix = 0;
119     for (ModelPerformer performer : performers) {
120         int keyFrom = performer.getKeyFrom();
121         int keyTo = performer.getKeyTo();
122         int velFrom = performer.getVelFrom();

```

```

123     int velTo = performer.getVelTo();
124     if (keyFrom > keyTo) continue;
125     if (velFrom > velTo) continue;
126     keyFrom = restrict(keyFrom);
127     keyTo = restrict(keyTo);
128     velFrom = restrict(velFrom);
129     velTo = restrict(velTo);
130     int x_from = trantables[0][keyFrom];
131     int x_to = trantables[0][keyTo + 1];
132     int y_from = trantables[1][velFrom];
133     int y_to = trantables[1][velTo + 1];
134     if (x_to == -1)
135         x_to = counters[0];
136     if (y_to == -1)
137         y_to = counters[1];
138     for (int y = y_from; y < y_to; y++) {
139         int i = x_from + y * counters[0];
140         for (int x = x_from; x < x_to; x++) {
141             int[] mprev = mat[i];
142             if (mprev == null) {
143                 mat[i] = new int[] { ix };
144             } else {
145                 int[] mnew = new int[mprev.length + 1];
146                 mnew[mnew.length - 1] = ix;
147                 for (int k = 0; k < mprev.length; k++)
148                     mnew[k] = mprev[k];
149                 mat[i] = mnew;
150             }
151             i++;
152         }
153     }
154     ix++;
155 }
156
157
158 public void close() {
159 }
160
161 public void noteOff(int noteNumber, int velocity) {
162     if (!noteOffUsed)
163         return;
164     int[] plist = lookupIndex(noteNumber, velocity);
165     if (plist == null) return;
166     for (int i : plist) {
167         ModelPerformer p = performers[i];
168         if (p.isReleaseTriggered()) {
169             player.play(i, null);
170         }
171     }
172 }
173
174 public void noteOn(int noteNumber, int velocity) {
175     if (!noteOnUsed)
176         return;
177     int[] plist = lookupIndex(noteNumber, velocity);
178     if (plist == null) return;
179     for (int i : plist) {
180         ModelPerformer p = performers[i];
181         if (!p.isReleaseTriggered()) {
182             player.play(i, null);
183         }
184     }

```

185 }
186 }

57 com/sun/media/sound/ModelStandardTransform.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 /**
28 * A standard transformer used in connection blocks.
29 * It expects input values to be between 0 and 1.
30 *
31 * The result of the transform is
32 *   between 0 and 1 if polarity = unipolar and
33 *   between -1 and 1 if polarity = bipolar.
34 *
35 * These constraints only applies to Concave, Convex and Switch transforms.
36 *
37 * @author Karl Helgason
38 */
39 public class ModelStandardTransform implements ModelTransform {
40
41     public static final boolean DIRECTION_MIN2MAX = false;
42     public static final boolean DIRECTION_MAX2MIN = true;
43     public static final boolean POLARITY_UNIPOLAR = false;
44     public static final boolean POLARITY_BIPOLAR = true;
45     public static final int TRANSFORM_LINEAR = 0;
46     // concave: output = (20*log10(127^2/value^2)) / 96
47     public static final int TRANSFORM_CONCAVE = 1;
48     // convex: same as concave except that start and end point are reversed.
49     public static final int TRANSFORM_CONVEX = 2;
50     // switch: if value > avg(max,min) then max else min
51     public static final int TRANSFORM_SWITCH = 3;
52     public static final int TRANSFORM_ABSOLUTE = 4;
53     private boolean direction = DIRECTION_MIN2MAX;
54     private boolean polarity = POLARITY_UNIPOLAR;
55     private int transform = TRANSFORM_LINEAR;
56
57     public ModelStandardTransform() {
58     }
59
60     public ModelStandardTransform(boolean direction) {
```



```

61     this.direction = direction;
62 }
63
64 public ModelStandardTransform(boolean direction, boolean polarity) {
65     this.direction = direction;
66     this.polarity = polarity;
67 }
68
69 public ModelStandardTransform(boolean direction, boolean polarity,
70     int transform) {
71     this.direction = direction;
72     this.polarity = polarity;
73     this.transform = transform;
74 }
75
76 public double transform(double value) {
77     double s;
78     double a;
79     if (direction == DIRECTION_MAX2MIN)
80         value = 1.0 - value;
81     if (polarity == POLARITY_BIPOLAR)
82         value = value * 2.0 - 1.0;
83     switch (transform) {
84         case TRANSFORM_CONCAVE:
85             s = Math.signum(value);
86             a = Math.abs(value);
87             a = -((5.0 / 12.0) / Math.log(10)) * Math.log(1.0 - a);
88             if (a < 0)
89                 a = 0;
90             else if (a > 1)
91                 a = 1;
92             return s * a;
93         case TRANSFORM_CONVEX:
94             s = Math.signum(value);
95             a = Math.abs(value);
96             a = 1.0 + ((5.0 / 12.0) / Math.log(10)) * Math.log(a);
97             if (a < 0)
98                 a = 0;
99             else if (a > 1)
100                 a = 1;
101             return s * a;
102         case TRANSFORM_SWITCH:
103             if (polarity == POLARITY_BIPOLAR)
104                 return (value > 0) ? 1 : -1;
105             else
106                 return (value > 0.5) ? 1 : 0;
107         case TRANSFORM_ABSOLUTE:
108             return Math.abs(value);
109         default:
110             break;
111     }
112
113     return value;
114 }
115
116 public boolean getDirection() {
117     return direction;
118 }
119
120 public void setDirection(boolean direction) {
121     this.direction = direction;
122 }

```

```
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

```
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 /**
28 * Model transform interface.
29 *
30 * @author Karl Helgason
31 */
32 public interface ModelTransform {
33
34     abstract public double transform(double value);
35 }
```

59 com/sun/media/sound/ModelWavetable.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 /**
28 * This is a wavetable oscillator interface.
29 *
30 * @author Karl Helgason
31 */
32 public interface ModelWavetable extends ModelOscillator {
33
34     public static final int LOOP_TYPE_OFF = 0;
35     public static final int LOOP_TYPE_FORWARD = 1;
36     public static final int LOOP_TYPE_RELEASE = 2;
37     public static final int LOOP_TYPE_PINGPONG = 4;
38     public static final int LOOP_TYPE_REVERSE = 8;
39
40     public AudioFloatInputStream openStream();
41
42     public float getLoopLength();
43
44     public float getLoopStart();
45
46     public int getLoopType();
47
48     public float getPitchcorrection();
49 }
```

60 com/sun/media/sound/RIFFInvalidDataException.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 /**
28 * This exception is used when a RIFF file contains illegal or unexpected data.
29 *
30 * @author Karl Helgason
31 */
32 public class RIFFInvalidDataException extends InvalidDataException {
33
34     private static final long serialVersionUID = 1L;
35
36     public RIFFInvalidDataException() {
37         super("Invalid_Data!");
38     }
39
40     public RIFFInvalidDataException(String s) {
41         super(s);
42     }
43 }
```

61 com/sun/media/sound/RIFFInvalidFormatException.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 /**
28 * This exception is used when a reader is used to read RIFF file of a format it
29 * doesn't understand or support.
30 *
31 * @author Karl Helgason
32 */
33 public class RIFFInvalidFormatException extends InvalidFormatException {
34
35     private static final long serialVersionUID = 1L;
36
37     public RIFFInvalidFormatException() {
38         super("Invalid_format!");
39     }
40
41     public RIFFInvalidFormatException(String s) {
42         super(s);
43     }
44 }
```

62 com/sun/media/sound/RIFFReader.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.io.EOFException;
28 import java.io.IOException;
29 import java.io.InputStream;
30
31 /**
32 * Resource Interchange File Format (RIFF) stream decoder.
33 *
34 * @author Karl Helgason
35 */
36 public class RIFFReader extends InputStream {
37
38     private RIFFReader root;
39     private long filepointer = 0;
40     private String fourcc;
41     private String riff_type = null;
42     private long ckSize = 0;
43     private InputStream stream;
44     private long avail;
45     private RIFFReader lastiterator = null;
46
47     public RIFFReader(InputStream stream) throws IOException {
48
49         if (stream instanceof RIFFReader)
50             root = ((RIFFReader)stream).root;
51         else
52             root = this;
53
54         this.stream = stream;
55         avail = Integer.MAX_VALUE;
56         ckSize = Integer.MAX_VALUE;
57
58         // Check for RIFF null paddings,
59         int b;
60         while (true) {
```

```

61         b = read();
62         if (b == -1) {
63             fourcc = ""; // don't put null value into fourcc,
64             // because it is expected to
65             // always contain a string value
66             riff_type = null;
67             avail = 0;
68             return;
69         }
70         if (b != 0)
71             break;
72     }
73
74     byte[] fourcc = new byte[4];
75     fourcc[0] = (byte) b;
76     readFully(fourcc, 1, 3);
77     this.fourcc = new String(fourcc, "ascii");
78     ckSize = readUnsignedInt();
79
80     avail = this.ckSize;
81
82     if (getFormat().equals("RIFF") || getFormat().equals("LIST")) {
83         byte[] format = new byte[4];
84         readFully(format);
85         this.riff_type = new String(format, "ascii");
86     }
87 }
88
89 public long getFilePointer() throws IOException {
90     return root.filepointer;
91 }
92
93 public boolean hasNextChunk() throws IOException {
94     if (lastiterator != null)
95         lastiterator.finish();
96     return avail != 0;
97 }
98
99 public RIFFReader nextChunk() throws IOException {
100     if (lastiterator != null)
101         lastiterator.finish();
102     if (avail == 0)
103         return null;
104     lastiterator = new RIFFReader(this);
105     return lastiterator;
106 }
107
108 public String getFormat() {
109     return fourcc;
110 }
111
112 public String getType() {
113     return riff_type;
114 }
115
116 public long getSize() {
117     return ckSize;
118 }
119
120 public int read() throws IOException {
121     if (avail == 0)
122         return -1;

```



```

123     int b = stream.read();
124     if (b == -1)
125         return -1;
126     avail--;
127     filepointer++;
128     return b;
129 }
130
131 public int read(byte[] b, int offset, int len) throws IOException {
132     if (avail == 0)
133         return -1;
134     if (len > avail) {
135         int rlen = stream.read(b, offset, (int)avail);
136         if (rlen != -1)
137             filepointer += rlen;
138         avail = 0;
139         return rlen;
140     } else {
141         int ret = stream.read(b, offset, len);
142         if (ret == -1)
143             return -1;
144         avail -= ret;
145         filepointer += ret;
146         return ret;
147     }
148 }
149
150 public final void readFully(byte b[]) throws IOException {
151     readFully(b, 0, b.length);
152 }
153
154 public final void readFully(byte b[], int off, int len) throws IOException {
155     if (len < 0)
156         throw new IndexOutOfBoundsException();
157     while (len > 0) {
158         int s = read(b, off, len);
159         if (s < 0)
160             throw new EOFException();
161         if (s == 0)
162             Thread.yield();
163         off += s;
164         len -= s;
165     }
166 }
167
168 public final long skipBytes(long n) throws IOException {
169     if (n < 0)
170         return 0;
171     long skipped = 0;
172     while (skipped != n) {
173         long s = skip(n - skipped);
174         if (s < 0)
175             break;
176         if (s == 0)
177             Thread.yield();
178         skipped += s;
179     }
180     return skipped;
181 }
182
183 public long skip(long n) throws IOException {
184     if (avail == 0)

```

```

185         return -1;
186     if (n > avail) {
187         long len = stream.skip(avail);
188         if (len != -1)
189             filepointer += len;
190         avail = 0;
191         return len;
192     } else {
193         long ret = stream.skip(n);
194         if (ret == -1)
195             return -1;
196         avail -= ret;
197         filepointer += ret;
198         return ret;
199     }
200 }
201
202 public int available() {
203     return (int)avail;
204 }
205
206 public void finish() throws IOException {
207     if (avail != 0) {
208         skipBytes(avail);
209     }
210 }
211
212 // Read ASCII chars from stream
213 public String readString(int len) throws IOException {
214     byte[] buff = new byte[len];
215     readFully(buff);
216     for (int i = 0; i < buff.length; i++) {
217         if (buff[i] == 0) {
218             return new String(buff, 0, i, "ascii");
219         }
220     }
221     return new String(buff, "ascii");
222 }
223
224 // Read 8 bit signed integer from stream
225 public byte readByte() throws IOException {
226     int ch = read();
227     if (ch < 0)
228         throw new EOFException();
229     return (byte) ch;
230 }
231
232 // Read 16 bit signed integer from stream
233 public short readShort() throws IOException {
234     int ch1 = read();
235     int ch2 = read();
236     if (ch1 < 0)
237         throw new EOFException();
238     if (ch2 < 0)
239         throw new EOFException();
240     return (short)(ch1 | (ch2 << 8));
241 }
242
243 // Read 32 bit signed integer from stream
244 public int readInt() throws IOException {
245     int ch1 = read();
246     int ch2 = read();

```

```

247     int ch3 = read();
248     int ch4 = read();
249     if (ch1 < 0)
250         throw new EOFException();
251     if (ch2 < 0)
252         throw new EOFException();
253     if (ch3 < 0)
254         throw new EOFException();
255     if (ch4 < 0)
256         throw new EOFException();
257     return ch1 + (ch2 << 8) | (ch3 << 16) | (ch4 << 24);
258 }
259
260 // Read 64 bit signed integer from stream
261 public long readLong() throws IOException {
262     long ch1 = read();
263     long ch2 = read();
264     long ch3 = read();
265     long ch4 = read();
266     long ch5 = read();
267     long ch6 = read();
268     long ch7 = read();
269     long ch8 = read();
270     if (ch1 < 0)
271         throw new EOFException();
272     if (ch2 < 0)
273         throw new EOFException();
274     if (ch3 < 0)
275         throw new EOFException();
276     if (ch4 < 0)
277         throw new EOFException();
278     if (ch5 < 0)
279         throw new EOFException();
280     if (ch6 < 0)
281         throw new EOFException();
282     if (ch7 < 0)
283         throw new EOFException();
284     if (ch8 < 0)
285         throw new EOFException();
286     return ch1 | (ch2 << 8) | (ch3 << 16) | (ch4 << 24)
287         | (ch5 << 32) | (ch6 << 40) | (ch7 << 48) | (ch8 << 56);
288 }
289
290 // Read 8 bit unsigned integer from stream
291 public int readUnsignedByte() throws IOException {
292     int ch = read();
293     if (ch < 0)
294         throw new EOFException();
295     return ch;
296 }
297
298 // Read 16 bit unsigned integer from stream
299 public int readUnsignedShort() throws IOException {
300     int ch1 = read();
301     int ch2 = read();
302     if (ch1 < 0)
303         throw new EOFException();
304     if (ch2 < 0)
305         throw new EOFException();
306     return ch1 | (ch2 << 8);
307 }
308

```

```

309 // Read 32 bit unsigned integer from stream
310 public long readUnsignedInt() throws IOException {
311     long ch1 = read();
312     long ch2 = read();
313     long ch3 = read();
314     long ch4 = read();
315     if (ch1 < 0)
316         throw new EOFException();
317     if (ch2 < 0)
318         throw new EOFException();
319     if (ch3 < 0)
320         throw new EOFException();
321     if (ch4 < 0)
322         throw new EOFException();
323     return ch1 + (ch2 << 8) | (ch3 << 16) | (ch4 << 24);
324 }
325
326 public void close() throws IOException {
327     finish();
328     if (this == root)
329         stream.close();
330     stream = null;
331 }
332 }

```

63 com/sun/media/sound/RIFFWriter.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.io.File;
28 import java.io.FileNotFoundException;
29 import java.io.IOException;
30 import java.io.OutputStream;
31 import java.io.RandomAccessFile;
32
33 /**
34 * Resource Interchange File Format (RIFF) stream encoder.
35 *
36 * @author Karl Helgason
37 */
38 public class RIFFWriter extends OutputStream {
39
40     private interface RandomAccessWriter {
41
42         public void seek(long chunksizepointer) throws IOException;
43
44         public long getPointer() throws IOException;
45
46         public void close() throws IOException;
47
48         public void write(int b) throws IOException;
49
50         public void write(byte[] b, int off, int len) throws IOException;
51
52         public void write(byte[] bytes) throws IOException;
53
54         public long length() throws IOException;
55
56         public void setLength(long i) throws IOException;
57     }
58
59     private static class RandomAccessFileWriter implements RandomAccessWriter {
60
```

```

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)

```

```

185         throws IOException {
186     if (chunktype == 0)
187         if (raf.length() != 0)
188             raf.setLength(0);
189     this.raf = raf;
190     if (raf.getPointer() % 2 != 0)
191         raf.write(0);
192
193     if (chunktype == 0)
194         raf.write("RIFF".getBytes("ascii"));
195     else if (chunktype == 1)
196         raf.write("LIST".getBytes("ascii"));
197     else
198         raf.write((format + "_____").substring(0, 4).getBytes("ascii"));
199
200     chunksizepointer = raf.getPointer();
201     this.chunktype = 2;
202     writeUnsignedInt(0);
203     this.chunktype = chunktype;
204     startpointer = raf.getPointer();
205     if (chunktype != 2)
206         raf.write((format + "_____").substring(0, 4).getBytes("ascii"));
207
208 }
209
210 public void seek(long pos) throws IOException {
211     raf.seek(pos);
212 }
213
214 public long getFilePointer() throws IOException {
215     return raf.getPointer();
216 }
217
218 public void setWriteOverride(boolean writeoverride) {
219     this.writeoverride = writeoverride;
220 }
221
222 public boolean getWriteOverride() {
223     return writeoverride;
224 }
225
226 public void close() throws IOException {
227     if (!open)
228         return;
229     if (childchunk != null) {
230         childchunk.close();
231         childchunk = null;
232     }
233
234     int bakchunktype = chunktype;
235     long fpointer = raf.getPointer();
236     raf.seek(chunksizepointer);
237     chunktype = 2;
238     writeUnsignedInt(fpointer - startpointer);
239
240     if (bakchunktype == 0)
241         raf.close();
242     else
243         raf.seek(fpointer);
244     open = false;
245     raf = null;
246 }

```



```

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

```

```

309 public void writeString(String string, int len) throws IOException {
310     byte[] buff = string.getBytes();
311     if (buff.length > len)
312         write(buff, 0, len);
313     else {
314         write(buff);
315         for (int i = buff.length; i < len; i++)
316             write(0);
317     }
318 }
319
320 // Write 8 bit signed integer to stream
321 public void writeByte(int b) throws IOException {
322     write(b);
323 }
324
325 // Write 16 bit signed integer to stream
326 public void writeShort(short b) throws IOException {
327     write((b >>> 0) & 0xFF);
328     write((b >>> 8) & 0xFF);
329 }
330
331 // Write 32 bit signed integer to stream
332 public void writeInt(int b) throws IOException {
333     write((b >>> 0) & 0xFF);
334     write((b >>> 8) & 0xFF);
335     write((b >>> 16) & 0xFF);
336     write((b >>> 24) & 0xFF);
337 }
338
339 // Write 64 bit signed integer to stream
340 public void writeLong(long b) throws IOException {
341     write((int) (b >>> 0) & 0xFF);
342     write((int) (b >>> 8) & 0xFF);
343     write((int) (b >>> 16) & 0xFF);
344     write((int) (b >>> 24) & 0xFF);
345     write((int) (b >>> 32) & 0xFF);
346     write((int) (b >>> 40) & 0xFF);
347     write((int) (b >>> 48) & 0xFF);
348     write((int) (b >>> 56) & 0xFF);
349 }
350
351 // Write 8 bit unsigned integer to stream
352 public void writeUnsignedByte(int b) throws IOException {
353     writeByte((byte) b);
354 }
355
356 // Write 16 bit unsigned integer to stream
357 public void writeUnsignedShort(int b) throws IOException {
358     writeShort((short) b);
359 }
360
361 // Write 32 bit unsigned integer to stream
362 public void writeUnsignedInt(long b) throws IOException {
363     writeInt((int) b);
364 }
365 }

```

64 com/sun/media/sound/RealTimeSequencer.java

```
1  /*
2  * Copyright (c) 2003, 2011, Oracle and/or its affiliates. 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. Oracle designates this
8  * particular file as subject to the "Classpath" exception as provided
9  * by Oracle 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 Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
22 * or visit www.oracle.com if you need additional information or have any
23 * questions.
24 */
25
26 package com.sun.media.sound;
27
28 import java.io.ByteArrayOutputStream;
29 import java.io.ByteArrayInputStream;
30 import java.io.DataOutputStream;
31 import java.io.IOException;
32 import java.io.InputStream;
33
34 import java.util.ArrayList;
35 import java.util.List;
36
37 import javax.sound.midi.*;
38
39
40 /**
41  * A Real Time Sequencer
42  *
43  * @author Florian Bomers
44  */
45
46 /* TODO:
47  * - rename PlayThread to PlayEngine (because isn't a thread)
48  */
49 class RealTimeSequencer extends AbstractMidiDevice implements Sequencer, AutoConnectSequencer {
50
51     // STATIC VARIABLES
52
53     /** debugging flags */
54     private final static boolean DEBUG_PUMP = false;
55     private final static boolean DEBUG_PUMP_ALL = false;
56
57     /**
58      * Event Dispatcher thread. Should be using a shared event
59      * dispatcher instance with a factory in EventDispatcher
60      */
61 }
```

```

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 public synchronized void start() {
244     if (Printer.trace) Printer.trace(">>_RealTimeSequencer:_start()");
245
246     // sequencer not open: throw an exception
247     if (!isOpen()) {
248         throw new IllegalStateException("sequencer_not_open");
249     }
250
251     // sequence not available: throw an exception
252     if (sequence == null) {
253         throw new IllegalStateException("sequence_not_set");
254     }
255
256     // already running: return quietly
257     if (running == true) {
258         return;
259     }
260
261     // start playback
262     implStart();
263
264     if (Printer.trace) Printer.trace("<<_RealTimeSequencer:_start()_completed");
265 }
266
267
268 public synchronized void stop() {
269     if (Printer.trace) Printer.trace(">>_RealTimeSequencer:_stop()");
270
271     if (!isOpen()) {
272         throw new IllegalStateException("sequencer_not_open");
273     }
274     stopRecording();
275
276     // not running; just return
277     if (running == false) {
278         if (Printer.trace) Printer.trace("<<_RealTimeSequencer:_stop()_not_running!");
279         return;
280     }
281
282     // stop playback
283     implStop();
284
285     if (Printer.trace) Printer.trace("<<_RealTimeSequencer:_stop()_completed");
286 }
287
288
289 public boolean isRunning() {
290     return running;
291 }
292
293
294 public void startRecording() {
295     if (!isOpen()) {
296         throw new IllegalStateException("Sequencer_not_open");
297     }
298
299     start();
300     recording = true;
301 }
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()) {

```

```

429         cacheTempoFactor = factor;
430     } else {
431         getDataPump().setTempoFactor(factor);
432         // don't need cache anymore
433         cacheTempoFactor = -1;
434     }
435 }
436
437
438 public float getTempoFactor() {
439     if (Printer.trace) Printer.trace(">>_RealTimeSequencer:_getTempoFactor()_");
440
441     if (needCaching()) {
442         if (cacheTempoFactor != -1) {
443             return cacheTempoFactor;
444         }
445         return 1.0f;
446     }
447     return getDataPump().getTempoFactor();
448 }
449
450
451 public long getTickLength() {
452     if (Printer.trace) Printer.trace(">>_RealTimeSequencer:_getTickLength()_");
453
454     if (sequence == null) {
455         return 0;
456     }
457
458     return sequence.getTickLength();
459 }
460
461
462 public synchronized long getTickPosition() {
463     if (Printer.trace) Printer.trace(">>_RealTimeSequencer:_getTickPosition()_");
464
465     if (getDataPump() == null || sequence == null) {
466         return 0;
467     }
468
469     return getDataPump().getTickPos();
470 }
471
472
473 public synchronized void setTickPosition(long tick) {
474     if (tick < 0) {
475         // should throw IllegalArgumentException
476         return;
477     }
478
479     if (Printer.trace) Printer.trace(">>_RealTimeSequencer:_setTickPosition("+tick+"_");
480
481     if (getDataPump() == null) {
482         if (tick != 0) {
483             // throw new InvalidStateException("cannot set position in closed state");
484         }
485     }
486     else if (sequence == null) {
487         if (tick != 0) {
488             // throw new InvalidStateException("cannot set position if sequence is not set");
489         }
490     } else {

```

```

491         getDataPump().setTickPos(tick);
492     }
493 }
494
495
496 public long getMicrosecondLength() {
497     if (Printer.trace) Printer.trace(">>_RealTimeSequencer:_getMicrosecondLength()_");
498
499     if (sequence == null) {
500         return 0;
501     }
502
503     return sequence.getMicrosecondLength();
504 }
505
506
507 public long getMicrosecondPosition() {
508     if (Printer.trace) Printer.trace(">>_RealTimeSequencer:_getMicrosecondPosition()_");
509
510     if (getDataPump() == null || sequence == null) {
511         return 0;
512     }
513     synchronized (tempoCache) {
514         return MidiUtils.tick2microsecond(sequence, getDataPump().getTickPos(), tempoCache);
515     }
516 }
517
518
519 public void setMicrosecondPosition(long microseconds) {
520     if (microseconds < 0) {
521         // should throw IllegalArgumentException
522         return;
523     }
524
525     if (Printer.trace) Printer.trace(">>_RealTimeSequencer:_setMicrosecondPosition(\"+
526         microseconds+\")_");
527
528     if (getDataPump() == null) {
529         if (microseconds != 0) {
530             // throw new IllegalStateException("cannot set position in closed state");
531         }
532     }
533     else if (sequence == null) {
534         if (microseconds != 0) {
535             // throw new IllegalStateException("cannot set position if sequence is not set");
536         }
537     }
538     else {
539         synchronized(tempoCache) {
540             setTickPosition(MidiUtils.microsecond2tick(sequence, microseconds, tempoCache));
541         }
542     }
543 }
544
545 public void setMasterSyncMode(Sequencer.SyncMode sync) {
546     // not supported
547 }
548
549 public Sequencer.SyncMode getMasterSyncMode() {
550     return masterSyncMode;
551 }

```

```

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     {
647         synchronized(controllerEventListeners) {
648
649             // first find the listener.  if we have one, add the controllers
650             // if not, create a new element for it.
651             ControllerListElement cve = null;
652             boolean flag = false;
653             for(int i=0; i < controllerEventListeners.size(); i++) {
654
655                 cve = (ControllerListElement) controllerEventListeners.get(i);
656
657                 if (cve.listener.equals(listener)) {
658                     cve.addControllers(controllers);
659                     flag = true;
660                     break;
661                 }
662             }
663             if (!flag) {
664                 cve = new ControllerListElement(listener, controllers);
665                 controllerEventListeners.add(cve);
666             }
667
668             // and return all the controllers this listener is interested in
669             return cve.getControllers();
670         }
671     }
672
673     public int[] removeControllerEventListener(ControllerEventListener listener, int[]
        controllers) {

```

```

674     synchronized(controllerEventListeners) {
675         ControllerListElement cve = null;
676         boolean flag = false;
677         for (int i=0; i < controllerEventListeners.size(); i++) {
678             cve = (ControllerListElement) controllerEventListeners.get(i);
679             if (cve.listener.equals(listener)) {
680                 cve.removeControllers(controllers);
681                 flag = true;
682                 break;
683             }
684         }
685         if (!flag) {
686             return new int[0];
687         }
688         if (controllers == null) {
689             int index = controllerEventListeners.indexOf(cve);
690             if (index >= 0) {
691                 controllerEventListeners.remove(index);
692             }
693             return new int[0];
694         }
695         return cve.getControllers();
696     }
697 }
698
699
700 //////////////// LOOPING (added in 1.5) ///////////////////
701
702 public void setLoopStartPoint(long tick) {
703     if ((tick > getTickLength())
704         || ((loopEnd != -1) && (tick > loopEnd))
705         || (tick < 0)) {
706         throw new IllegalArgumentException("invalid_loop_start_point:"+tick);
707     }
708     loopStart = tick;
709 }
710
711 public long getLoopStartPoint() {
712     return loopStart;
713 }
714
715 public void setLoopEndPoint(long tick) {
716     if ((tick > getTickLength())
717         || ((loopStart > tick) && (tick != -1))
718         || (tick < -1)) {
719         throw new IllegalArgumentException("invalid_loop_end_point:"+tick);
720     }
721     loopEnd = tick;
722 }
723
724 public long getLoopEndPoint() {
725     return loopEnd;
726 }
727
728 public void setLoopCount(int count) {
729     if (count != LOOP_CONTINUOUSLY
730         && count < 0) {
731         throw new IllegalArgumentException("illegal_value_for_loop_count:"+count);
732     }
733     loopCount = count;
734     if (getDataPump() != null) {
735         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                 ()));
824         } else {
825             setTempoInMPQ((float) cacheTempoMPQ);
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_
842                 not_instanciated!");
843         } else {
844             // Interrupt playback loop.
845             playThread.close();
846             playThread = null;
847         }
848
849         super.implClose();
850
851         sequence = null;
852         running = false;
853         cacheTempoMPQ = -1;
854         cacheTempoFactor = -1;
855         trackMuted = null;
856         trackSolo = null;
857         loopStart = 0;
858         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_
880             not_instanciated!");
881         return;
882     }
883
884     tempoCache.refresh(sequence);
885     if (!running) {
886         running = true;
887         playThread.start();
888     }
889     if (Printer.trace) Printer.trace("<<_RealTimeSequencer:_implStart()_completed");
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
897             _instanciated!");
898         return;
899     }
900
901     recording = false;
902     if (running) {
903         running = false;
904         playThread.stop();
905     }
906     if (Printer.trace) Printer.trace("<<_RealTimeSequencer:_implStop()_completed");
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);

```

```

918     }
919
920     /**
921      * Send midi player events.
922      */
923     protected void sendControllerEvents(MidiMessage message) {
924         int size = controllerEventListeners.size();
925         if (size == 0) return;
926
927         //if (Printer.debug) Printer.debug("sending a controller event");
928
929         if (! (message instanceof ShortMessage)) {
930             if (Printer.debug) Printer.debug("sendControllerEvents:_message_is_NOT_instanceof_
ShortMessage!");
931             return;
932         }
933         ShortMessage msg = (ShortMessage) message;
934         int controller = msg.getData1();
935         List sendToListeners = new ArrayList();
936         for (int i = 0; i < size; i++) {
937             ControllerListElement cve = (ControllerListElement) controllerEventListeners.get(i);
938             for(int j = 0; j < cve.controllers.length; j++) {
939                 if (cve.controllers[j] == controller) {
940                     sendToListeners.add(cve.listener);
941                     break;
942                 }
943             }
944         }
945         eventDispatcher.sendAudioEvents(message, sendToListeners);
946     }
947
948
949
950     private boolean needCaching() {
951         return !isOpen() || (sequence == null) || (playThread == null);
952     }
953
954     /**
955      * return the data pump instance, owned by play thread
956      * if playthread is null, return null.
957      * This method is guaranteed to return non-null if
958      * needCaching returns false
959      */
960     private DataPump getDataPump() {
961         if (playThread != null) {
962             return playThread.getDataPump();
963         }
964         return null;
965     }
966
967     private MidiUtils.TempoCache getTempoCache() {
968         return tempoCache;
969     }
970
971     private static boolean[] ensureBoolArraySize(boolean[] array, int desiredSize) {
972         if (array == null) {
973             return new boolean[desiredSize];
974         }
975         if (array.length < desiredSize) {
976             boolean[] newArray = new boolean[desiredSize];
977             System.arraycopy(array, 0, newArray, 0, array.length);
978             return newArray;

```

```

979     }
980     return array;
981 }
982
983
984 // OVERRIDES OF ABSTRACT MIDI DEVICE METHODS
985
986 protected boolean hasReceivers() {
987     return true;
988 }
989
990 // for recording
991 protected Receiver createReceiver() throws MidiUnavailableException {
992     return new SequencerReceiver();
993 }
994
995
996 protected boolean hasTransmitters() {
997     return true;
998 }
999
1000
1001 protected Transmitter createTransmitter() throws MidiUnavailableException {
1002     return new SequencerTransmitter();
1003 }
1004
1005
1006 // interface AutoConnectSequencer
1007 public void setAutoConnect(Receiver autoConnectedReceiver) {
1008     this.autoConnect = (autoConnectedReceiver != null);
1009     this.autoConnectedReceiver = autoConnectedReceiver;
1010 }
1011
1012
1013
1014 // INNER CLASSES
1015
1016 /**
1017  * An own class to distinguish the class name from
1018  * the transmitter of other devices
1019  */
1020 private class SequencerTransmitter extends BasicTransmitter {
1021     private SequencerTransmitter() {
1022         super();
1023     }
1024 }
1025
1026
1027 class SequencerReceiver extends AbstractReceiver {
1028
1029     protected void implSend(MidiMessage message, long timeStamp) {
1030         if (recording) {
1031             long tickPos = 0;
1032
1033             // convert timeStamp to ticks
1034             if (timeStamp < 0) {
1035                 tickPos = getTickPosition();
1036             } else {
1037                 synchronized(tempoCache) {
1038                     tickPos = MidiUtils.microsecond2tick(sequence, timeStamp, tempoCache);
1039                 }
1040             }

```

```

1041 // and record to the first matching Track
1042 Track track = null;
1043 // do not record real-time events
1044 // see 5048381: NullPointerException when saving a MIDI sequence
1045 if (message.getLength() > 1) {
1046     if (message instanceof ShortMessage) {
1047         ShortMessage sm = (ShortMessage) message;
1048         // all real-time messages have 0xF in the high nibble of the status byte
1049         if ((sm.getStatus() & 0xF0) != 0xF0) {
1050             track = RecordingTrack.get(recordingTracks, sm.getChannel());
1051         }
1052     } else {
1053         // $$jb: where to record meta, sysex events?
1054         // $$fb: the first recording track
1055         track = RecordingTrack.get(recordingTracks, -1);
1056     }
1057     if (track != null) {
1058         // create a copy of this message
1059         if (message instanceof ShortMessage) {
1060             message = new FastShortMessage((ShortMessage) message);
1061         } else {
1062             message = (MidiMessage) message.clone();
1063         }
1064
1065         // create new MidiEvent
1066         MidiEvent me = new MidiEvent(message, tickPos);
1067         track.add(me);
1068     }
1069 }
1070 }
1071 }
1072 }
1073 }
1074
1075 private static class RealTimeSequencerInfo extends MidiDevice.Info {
1076
1077     private static final String name = "Real_Time_Sequencer";
1078     private static final String vendor = "Oracle_Corporation";
1079     private static final String description = "Software_sequencer";
1080     private static final String version = "Version_1.0";
1081
1082     private RealTimeSequencerInfo() {
1083         super(name, vendor, description, version);
1084     }
1085 } // class Info
1086
1087 private class ControllerListElement {
1088
1089     // $$jb: using an array for controllers b/c its
1090     //         easier to deal with than turning all the
1091     //         ints into objects to use a Vector
1092     int [] controllers;
1093     ControllerEventListener listener;
1094
1095     private ControllerListElement(ControllerEventListener listener, int[] controllers) {
1096
1097         this.listener = listener;
1098         if (controllers == null) {
1099             controllers = new int[128];
1100             for (int i = 0; i < 128; i++) {

```

```

1103         controllers[i] = i;
1104     }
1105 }
1106 this.controllers = controllers;
1107 }
1108
1109 private void addControllers(int[] c) {
1110
1111     if (c==null) {
1112         controllers = new int[128];
1113         for (int i = 0; i < 128; i++) {
1114             controllers[i] = i;
1115         }
1116         return;
1117     }
1118     int temp[] = new int[ controllers.length + c.length ];
1119     int elements;
1120
1121     // first add what we have
1122     for(int i=0; i<controllers.length; i++) {
1123         temp[i] = controllers[i];
1124     }
1125     elements = controllers.length;
1126     // now add the new controllers only if we don't already have them
1127     for(int i=0; i<c.length; i++) {
1128         boolean flag = false;
1129
1130         for(int j=0; j<controllers.length; j++) {
1131             if (c[i] == controllers[j]) {
1132                 flag = true;
1133                 break;
1134             }
1135         }
1136         if (!flag) {
1137             temp[elements++] = c[i];
1138         }
1139     }
1140     // now keep only the elements we need
1141     int newc[] = new int[ elements ];
1142     for(int i=0; i<elements; i++){
1143         newc[i] = temp[i];
1144     }
1145     controllers = newc;
1146 }
1147
1148 private void removeControllers(int[] c) {
1149
1150     if (c==null) {
1151         controllers = new int[0];
1152     } else {
1153         int temp[] = new int[ controllers.length ];
1154         int elements = 0;
1155
1156
1157         for(int i=0; i<controllers.length; i++){
1158             boolean flag = false;
1159             for(int j=0; j<c.length; j++) {
1160                 if (controllers[i] == c[j]) {
1161                     flag = true;
1162                     break;
1163                 }
1164             }

```

```

1165         if (!flag){
1166             temp[elements++] = controllers[i];
1167         }
1168     }
1169     // now keep only the elements remaining
1170     int newc[] = new int[ elements ];
1171     for(int i=0; i<elements; i++) {
1172         newc[i] = temp[i];
1173     }
1174     controllers = newc;
1175 }
1176 }
1177 }
1178
1179 private int[] getControllers() {
1180
1181     // return a copy of our array of controllers,
1182     // so others can't mess with it
1183     if (controllers == null) {
1184         return null;
1185     }
1186
1187     int c[] = new int[controllers.length];
1188
1189     for(int i=0; i<controllers.length; i++){
1190         c[i] = controllers[i];
1191     }
1192     return c;
1193 }
1194
1195 } // class ControllerListElement
1196
1197
1198 static class RecordingTrack {
1199
1200     private Track track;
1201     private int channel;
1202
1203     RecordingTrack(Track track, int channel) {
1204         this.track = track;
1205         this.channel = channel;
1206     }
1207
1208     static RecordingTrack get(List recordingTracks, Track track) {
1209
1210         synchronized(recordingTracks) {
1211             int size = recordingTracks.size();
1212
1213             for (int i = 0; i < size; i++) {
1214                 RecordingTrack current = (RecordingTrack)recordingTracks.get(i);
1215                 if (current.track == track) {
1216                     return current;
1217                 }
1218             }
1219         }
1220         return null;
1221     }
1222
1223     static Track get(List recordingTracks, int channel) {
1224
1225         synchronized(recordingTracks) {
1226             int size = recordingTracks.size();

```

```

1227         for (int i = 0; i < size; i++) {
1228             RecordingTrack current = (RecordingTrack)recordingTracks.get(i);
1229             if ((current.channel == channel) || (current.channel == -1)) {
1230                 return current.track;
1231             }
1232         }
1233     }
1234     return null;
1235 }
1236 }
1237 }
1238
1239
1240 class PlayThread implements Runnable {
1241     private Thread thread;
1242     private Object lock = new Object();
1243
1244     /** true if playback is interrupted (in close) */
1245     boolean interrupted = false;
1246     boolean isPumping = false;
1247
1248     private DataPump dataPump = new DataPump();
1249
1250
1251     PlayThread() {
1252         // nearly MAX_PRIORITY
1253         int priority = Thread.NORM_PRIORITY
1254             + ((Thread.MAX_PRIORITY - Thread.NORM_PRIORITY) * 3) / 4;
1255         thread = JSSecurityManager.createThread(this,
1256             "Java_Sound_Sequencer", // name
1257             false, // daemon
1258             priority, // priority
1259             true); // doStart
1260     }
1261
1262     DataPump getDataPump() {
1263         return dataPump;
1264     }
1265
1266     synchronized void setSequence(Sequence seq) {
1267         dataPump.setSequence(seq);
1268     }
1269
1270
1271     /** start thread and pump. Requires up-to-date tempoCache */
1272     synchronized void start() {
1273         // mark the sequencer running
1274         running = true;
1275
1276         if (!dataPump.hasCachedTempo()) {
1277             long tickPos = getTickPosition();
1278             dataPump.setTempoMPQ(tempoCache.getTempoMPQAt(tickPos));
1279         }
1280         dataPump.checkPointMillis = 0; // means restarted
1281         dataPump.clearNoteOnCache();
1282         dataPump.needReindex = true;
1283
1284         dataPump.resetLoopCount();
1285
1286         // notify the thread
1287         synchronized(lock) {
1288             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  *

```



```

1350     * Make sure to NOT synchronize on RealTimeSequencer
1351     * anywhere here (even implicit). That is a sure deadlock!
1352     */
1353     public void run() {
1354
1355         while (!interrupted) {
1356             boolean EOM = false;
1357             boolean wasRunning = running;
1358             isPumping = !interrupted && running;
1359             while (!EOM && !interrupted && running) {
1360                 EOM = dataPump.pump();
1361
1362                 try {
1363                     Thread.sleep(1);
1364                 } catch (InterruptedException ie) {
1365                     // ignore
1366                 }
1367             }
1368             if (Printer.debug) {
1369                 Printer.debug("Exited_main_pump_loop_because:");
1370                 if (EOM) Printer.debug("_->_EOM_is_reached");
1371                 if (!running) Printer.debug("_->_running_was_set_to_false");
1372                 if (interrupted) Printer.debug("_->_interrupted_was_set_to_true");
1373             }
1374
1375             playThreadImplStop();
1376             if (wasRunning) {
1377                 dataPump.notesOff(true);
1378             }
1379             if (EOM) {
1380                 dataPump.setTickPos(sequence.getTickLength());
1381
1382                 // send EOT event (mis-used for end of media)
1383                 MetaMessage message = new MetaMessage();
1384                 try {
1385                     message.setMessage(MidiUtils.META_END_OF_TRACK_TYPE, new byte[0], 0);
1386                 } catch (InvalidMidiDataException e1) {}
1387                 sendMetaEvents(message);
1388             }
1389             synchronized (lock) {
1390                 isPumping = false;
1391                 // wake up a waiting stop() method
1392                 lock.notifyAll();
1393                 while (!running && !interrupted) {
1394                     try {
1395                         lock.wait();
1396                     } catch (Exception ex) {}
1397                 }
1398             }
1399             } // end of while(!EOM && !interrupted && running)
1400             if (Printer.debug) Printer.debug("end_of_play_thread");
1401         }
1402     }
1403
1404
1405     /**
1406     * class that does the actual dispatching of events,
1407     * used to be in native in MMAPI
1408     */
1409     private class DataPump {
1410         private float currTempo;           // MPQ tempo
1411         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 checkpoint
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 = lastTick;
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     }

```

```

1473         return ignoreTempoEventAt >= 0;
1474     }
1475
1476     // this method is also used internally in the pump!
1477     synchronized void setTempoMPQ(float tempoMPQ) {
1478         if (tempoMPQ > 0 && tempoMPQ != currTempo) {
1479             ignoreTempoEventAt = lastTick;
1480             this.currTempo = tempoMPQ;
1481             // re-calculate check point
1482             checkPointMillis = 0;
1483         }
1484     }
1485
1486     float getTempoMPQ() {
1487         return currTempo;
1488     }
1489
1490     synchronized void setTempoFactor(float factor) {
1491         if (factor > 0 && factor != this.tempoFactor) {
1492             tempoFactor = factor;
1493             inverseTempoFactor = 1.0f / factor;
1494             // re-calculate check point
1495             checkPointMillis = 0;
1496         }
1497     }
1498
1499     float getTempoFactor() {
1500         return tempoFactor;
1501     }
1502
1503     synchronized void muteSoloChanged() {
1504         boolean[] newDisabled = makeDisabledArray();
1505         if (running) {
1506             applyDisabledTracks(trackDisabled, newDisabled);
1507         }
1508         trackDisabled = newDisabled;
1509     }
1510
1511
1512
1513     synchronized void setSequence(Sequence seq) {
1514         if (seq == null) {
1515             init();
1516             return;
1517         }
1518         tracks = seq.getTracks();
1519         muteSoloChanged();
1520         resolution = seq.getResolution();
1521         divisionType = seq.getDivisionType();
1522         trackReadPos = new int[tracks.length];
1523         // trigger re-initialization
1524         checkPointMillis = 0;
1525         needReindex = true;
1526     }
1527
1528     synchronized void resetLoopCount() {
1529         currLoopCounter = loopCount;
1530     }
1531
1532     void clearNoteOnCache() {
1533         for (int i = 0; i < 128; i++) {
1534             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),
1547                     -1);
1548                 done++;
1549             }
1550         }
1551         /* all notes off */
1552         getTransmitterList().sendMessage((ShortMessage.CONTROL_CHANGE | ch) | (123<<8),
1553             -1);
1554         /* sustain off */
1555         getTransmitterList().sendMessage((ShortMessage.CONTROL_CHANGE | ch) | (64<<8),
1556             -1);
1557         if (doControllers) {
1558             /* reset all controllers */
1559             getTransmitterList().sendMessage((ShortMessage.CONTROL_CHANGE | ch) |
1560                 (121<<8), -1);
1561             done++;
1562         }
1563     }
1564     if (DEBUG_PUMP) Printer.println("_noteOff:_sent_"+done+"_messages.");
1565 }
1566
1567 private boolean[] makeDisabledArray() {
1568     if (tracks == null) {
1569         return null;
1570     }
1571     boolean[] newTrackDisabled = new boolean[tracks.length];
1572     boolean[] solo;
1573     boolean[] mute;
1574     synchronized(RealTimeSequencer.this) {
1575         mute = trackMuted;
1576         solo = trackSolo;
1577     }
1578     // if one track is solo, then only play solo
1579     boolean hasSolo = false;
1580     if (solo != null) {
1581         for (int i = 0; i < solo.length; i++) {
1582             if (solo[i]) {
1583                 hasSolo = true;
1584                 break;
1585             }
1586         }
1587     }
1588     if (hasSolo) {
1589         // only the channels with solo play, regardless of mute
1590         for (int i = 0; i < newTrackDisabled.length; i++) {
1591             newTrackDisabled[i] = (i >= solo.length) || (!solo[i]);
1592         }
1593     } else {
1594         // mute the selected channels
1595         for (int i = 0; i < newTrackDisabled.length; i++) {

```

```

1593         newTrackDisabled[i] = (mute != null) && (i < mute.length) &&(mute[i]);
1594     }
1595 }
1596 return newTrackDisabled;
1597 }
1598
1599 /**
1600  * chase all events from beginning of Track
1601  * and send note off for those events that are active
1602  * in noteOnCache array.
1603  * It is possible, of course, to catch notes from other tracks,
1604  * but better than more complicated logic to detect
1605  * which notes are really from this track
1606  */
1607 private void sendNoteOffIfOn(Track track, long endTick) {
1608     int size = track.size();
1609     int done = 0;
1610     try {
1611         for (int i = 0; i < size; i++) {
1612             MidiEvent event = track.get(i);
1613             if (event.getTick() > endTick) break;
1614             MidiMessage msg = event.getMessage();
1615             int status = msg.getStatus();
1616             int len = msg.getLength();
1617             if (len == 3 && ((status & 0xF0) == ShortMessage.NOTE_ON)) {
1618                 int note = -1;
1619                 if (msg instanceof ShortMessage) {
1620                     ShortMessage smsg = (ShortMessage) msg;
1621                     if (smsg.getData2() > 0) {
1622                         // only consider Note On with velocity > 0
1623                         note = smsg.getData1();
1624                     }
1625                 } else {
1626                     byte[] data = msg.getMessage();
1627                     if ((data[2] & 0x7F) > 0) {
1628                         // only consider Note On with velocity > 0
1629                         note = data[1] & 0x7F;
1630                     }
1631                 }
1632                 if (note >= 0) {
1633                     int bit = 1<<(status & 0x0F);
1634                     if ((noteOnCache[note] & bit) != 0) {
1635                         // the bit is set. Send Note Off
1636                         getTransmitterList().sendMessage(status | (note<<8), -1);
1637                         // clear the bit
1638                         noteOnCache[note] &= (0xFFFF ^ bit);
1639                         done++;
1640                     }
1641                 }
1642             }
1643         }
1644     } catch (ArrayIndexOutOfBoundsException aioobe) {
1645         // this happens when messages are removed
1646         // from the track while this method executes
1647     }
1648     if (DEBUG_PUMP) Printer.println("_sendNoteOffIfOn:_sent_" + done + "_messages.");
1649 }
1650
1651 /**
1652  * Runtime application of mute/solo:
1653  * 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);

```

```

1717     if (event.getTick() >= endTick) {
1718         if (doReindex && (trackNum < trackReadPos.length)) {
1719             trackReadPos[trackNum] = (i > 0)?(i-1):0;
1720             if (DEBUG_PUMP) Printer.println("chaseEvents: setting trackReadPos["
                +trackNum+"] = " +trackReadPos[trackNum]);
1721         }
1722         break;
1723     }
1724     MidiMessage msg = event.getMessage();
1725     int status = msg.getStatus();
1726     int len = msg.getLength();
1727     if (len == 3 && ((status & 0xF0) == ShortMessage.CONTROL_CHANGE)) {
1728         if (msg instanceof ShortMessage) {
1729             ShortMessage smsg = (ShortMessage) msg;
1730             tempArray[smsg.getData1() & 0x7F][status & 0x0F] = (byte) smsg.
                getData2();
1731         } else {
1732             byte[] data = msg.getMessage();
1733             tempArray[data[1] & 0x7F][status & 0x0F] = data[2];
1734         }
1735     }
1736     if (len == 2 && ((status & 0xF0) == ShortMessage.PROGRAM_CHANGE)) {
1737         if (msg instanceof ShortMessage) {
1738             ShortMessage smsg = (ShortMessage) msg;
1739             progs[status & 0x0F] = (byte) smsg.getData1();
1740         } else {
1741             byte[] data = msg.getMessage();
1742             progs[status & 0x0F] = data[1];
1743         }
1744     }
1745 }
1746 } catch (ArrayIndexOutOfBoundsException aioobe) {
1747     // this happens when messages are removed
1748     // from the track while this method executes
1749 }
1750 int numControllersSent = 0;
1751 // now send out the aggregated controllers and program changes
1752 for (int ch = 0; ch < 16; ch++) {
1753     for (int co = 0; co < 128; co++) {
1754         byte controllerValue = tempArray[co][ch];
1755         if (controllerValue >= 0) {
1756             int packedMsg = (ShortMessage.CONTROL_CHANGE | ch) | (co<<8) | (
                controllerValue<<16);
1757             getTransmitterList().sendMessage(packedMsg, -1);
1758             numControllersSent++;
1759         }
1760     }
1761     // send program change *after* controllers, to
1762     // correctly initialize banks
1763     if (progs[ch] >= 0) {
1764         getTransmitterList().sendMessage((ShortMessage.PROGRAM_CHANGE | ch) | (progs[
            ch]<<8), -1);
1765     }
1766     if (progs[ch] >= 0 || startTick == 0 || endTick == 0) {
1767         // reset pitch bend on this channel (E0 00 40)
1768         getTransmitterList().sendMessage((ShortMessage.PITCH_BEND | ch) | (0x40 <<
            16), -1);
1769         // reset sustain pedal on this channel
1770         getTransmitterList().sendMessage((ShortMessage.CONTROL_CHANGE | ch) | (64 <<
            8), -1);
1771     }
1772 }

```

```

1773         if (DEBUG_PUMP) Printer.println("_chaseTrackEvents_tick_" + trackNum + ":_sent_" +
1774             numControllersSent + "_controllers.");
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
1780             - 1));
1781         byte[][] tempArray = new byte[128][16];
1782         for (int t = 0; t < tracks.length; t++) {
1783             if ((trackDisabled == null)
1784                 || (trackDisabled.length <= t)
1785                 || (!trackDisabled[t])) {
1786                 // if track is not disabled, chase the events for it
1787                 chaseTrackEvents(t, startTick, endTick, true, tempArray);
1788             }
1789             if (DEBUG_PUMP) Printer.println("<<_chaseEvents");
1790         }
1791
1792         // playback related methods (pumping)
1793
1794         private long getCurrentTimeMillis() {
1795             return System.nanoTime() / 1000000L;
1796             //return perf.highResCounter() * 1000 / perfFreq;
1797         }
1798
1799         private long millis2tick(long millis) {
1800             if (divisionType != Sequence.PPQ) {
1801                 double dTick = (((double) millis) * tempoFactor)
1802                     * ((double) divisionType)
1803                     * ((double) resolution))
1804                     / ((double) 1000);
1805                 return (long) dTick;
1806             }
1807             return MidiUtils.microsec2ticks(millis * 1000,
1808                 currTempo * inverseTempoFactor,
1809                 resolution);
1810         }
1811
1812         private long tick2millis(long tick) {
1813             if (divisionType != Sequence.PPQ) {
1814                 double dMillis = (((double) tick) * 1000) /
1815                     (tempoFactor * ((double) divisionType) * ((double) resolution))
1816                     );
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 +
1828                     "]=_" + trackReadPos[trackNum]);
1829             }
1830         }

```



```

1831  /* returns if changes are pending */
1832  private boolean dispatchMessage(int trackNum, MidiEvent event) {
1833      boolean changesPending = false;
1834      MidiMessage message = event.getMessage();
1835      int msgStatus = message.getStatus();
1836      int msgLen = message.getLength();
1837      if (msgStatus == MetaMessage.META && msgLen >= 2) {
1838          // a meta message. Do not send it to the device.
1839          // 0xFF with length=1 is a MIDI realtime message
1840          // which shouldn't be in a Sequence, but we play it
1841          // nonetheless.
1842
1843          // see if this is a tempo message. Only on track 0.
1844          if (trackNum == 0) {
1845              int newTempo = MidiUtils.getTempoMPQ(message);
1846              if (newTempo > 0) {
1847                  if (event.getTick() != ignoreTempoEventAt) {
1848                      setTempoMPQ(newTempo); // sets ignoreTempoEventAt!
1849                      changesPending = true;
1850                  }
1851                  // next loop, do not ignore anymore tempo events.
1852                  ignoreTempoEventAt = -1;
1853              }
1854          }
1855          // send to listeners
1856          sendMetaEvents(message);
1857      } else {
1858          // not meta, send to device
1859          getTransmitterList().sendMessage(message, -1);
1860
1861          switch (msgStatus & 0xF0) {
1862              case ShortMessage.NOTE_OFF: {
1863                  // note off - clear the bit in the noteOnCache array
1864                  int note = ((ShortMessage) message).getData1() & 0x7F;
1865                  noteOnCache[note] &= (0xFFFF ^ (1<<(msgStatus & 0x0F)));
1866                  break;
1867              }
1868
1869              case ShortMessage.NOTE_ON: {
1870                  // note on
1871                  ShortMessage smsg = (ShortMessage) message;
1872                  int note = smsg.getData1() & 0x7F;
1873                  int vel = smsg.getData2() & 0x7F;
1874                  if (vel > 0) {
1875                      // if velocity > 0 set the bit in the noteOnCache array
1876                      noteOnCache[note] |= 1<<(msgStatus & 0x0F);
1877                  } else {
1878                      // if velocity = 0 clear the bit in the noteOnCache array
1879                      noteOnCache[note] &= (0xFFFF ^ (1<<(msgStatus & 0x0F)));
1880                  }
1881                  break;
1882              }
1883
1884              case ShortMessage.CONTROL_CHANGE: {
1885                  // if controller message, send controller listeners
1886                  sendControllerEvents(message);
1887                  break;
1888              }
1889          }
1890      }
1891      return changesPending;
1892  }

```

```

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._"
1915                 + "TargetTick="+targetTick);
1916             if (trackReadPos.length < tracks.length) {
1917                 trackReadPos = new int[tracks.length];
1918             }
1919             for (int t = 0; t < tracks.length; t++) {
1920                 ReindexTrack(t, targetTick);
1921                 if (DEBUG_PUMP_ALL) Printer.println("_Setting_trackReadPos["+t+"]="+
1922                     trackReadPos[t]);
1923             }
1924             needReindex = false;
1925             checkPointMillis = 0;
1926         }
1927
1928         // get target tick from current time in millis
1929         if (checkPointMillis == 0) {
1930             // new check point
1931             currMillis = getCurrentTimeMillis();
1932             checkPointMillis = currMillis;
1933             targetTick = lastTick;
1934             checkPointTick = targetTick;
1935             if (DEBUG_PUMP) Printer.println("New_checkpoint_to_"+currMillis+"_millis._"
1936                 + "TargetTick="+targetTick
1937                 + "_new_tempo="+MidiUtils.convertTempo(
1938                     currTempo)+"bpm");
1939         }
1940         else {
1941             // calculate current tick based on current time in milliseconds
1942             targetTick = checkPointTick + millis2tick(currMillis - checkPointMillis);
1943             if (DEBUG_PUMP_ALL) Printer.println("targetTick=_"+targetTick+"_at_"+
1944                 currMillis+"_millis");
1945             if ((loopEnd != -1)
1946                 && ((loopCount > 0 && currLoopCounter > 0)
1947                     || (loopCount == LOOP_CONTINUOUSLY))) {
1948                 if (lastTick <= loopEnd && targetTick >= loopEnd) {
1949                     // need to loop!
1950                     // only play until loop end
1951                     targetTick = loopEnd - 1;
1952                     doLoop = true;
1953                     if (DEBUG_PUMP) Printer.println("set_doLoop_to_true._lastTick="+
1954                         lastTick
1955                         + "_targetTick="+targetTick

```

```

1950         + "_loopEnd="+loopEnd
1951         + "_jumping_to_loopStart="+
            loopStart
1952         + "_new_currLoopCounter="+
            currLoopCounter);
1953         if (DEBUG_PUMP) Printer.println("_currMillis="+currMillis
1954         + "_checkPointMillis="+
            checkPointMillis
1955         + "_checkPointTick="+
            checkPointTick);
1956     }
1957 }
1958 }
1959 lastTick = targetTick;
1960 }
1961
1962 finishedTracks = 0;
1963
1964 for (int t = 0; t < tracks.length; t++) {
1965     try {
1966         boolean disabled = trackDisabled[t];
1967         Track thisTrack = tracks[t];
1968         int readPos = trackReadPos[t];
1969         int size = thisTrack.size();
1970         // play all events that are due until targetTick
1971         while (!changesPending && (readPos < size)
1972             && (currEvent = thisTrack.get(readPos)).getTick() <= targetTick) {
1973
1974             if ((readPos == size -1) && MidiUtils.isMetaEndOfTrack(currEvent.
1975                 getMessage())) {
1976                 // do not send out this message. Finished with this track
1977                 readPos = size;
1978                 break;
1979             }
1980             // TODO: some kind of heuristics if the MIDI messages have changed
1981             // significantly (i.e. deleted or inserted a bunch of messages)
1982             // since last time. Would need to set needReindex = true then
1983             readPos++;
1984             // only play this event if the track is enabled,
1985             // or if it is a tempo message on track 0
1986             // Note: cannot put this check outside
1987             // this inner loop in order to detect end of file
1988             if (!disabled ||
1989                 ((t == 0) && (MidiUtils.isMetaTempo(currEvent.getMessage())))) {
1990                 changesPending = dispatchMessage(t, currEvent);
1991             }
1992         }
1993         if (readPos >= size) {
1994             finishedTracks++;
1995         }
1996         if (DEBUG_PUMP_ALL) {
1997             System.out.print("_pumped_track_" + t + "(" + size + "_events)_"
1998                 + "_from_index_" + trackReadPos[t]
1999                 + "_to_" + (readPos - 1));
2000             System.out.print("_->_ticks:_");
2001             if (trackReadPos[t] < size) {
2002                 System.out.print("'" + (thisTrack.get(trackReadPos[t]).getTick()));
2003             } else {
2004                 System.out.print("EOT");
2005             }
2006             System.out.print("_to_");
2007             if (readPos < size) {

```

```

2007         System.out.print(""+(thisTrack.get(readPos-1).getTick()));
2008     } else {
2009         System.out.print("EOT");
2010     }
2011     System.out.println();
2012 }
2013 trackReadPos[t] = readPos;
2014 } catch (Exception e) {
2015     if (Printer.debug) Printer.debug("Exception_in_Sequencer_pump!");
2016     if (Printer.debug) e.printStackTrace();
2017     if (e instanceof ArrayIndexOutOfBoundsException) {
2018         needReindex = true;
2019         changesPending = true;
2020     }
2021 }
2022 if (changesPending) {
2023     break;
2024 }
2025 }
2026 EOM = (finishedTracks == tracks.length);
2027 if (doLoop
2028     || ( (loopCount > 0 && currLoopCounter > 0)
2029         || (loopCount == LOOP_CONTINUOUSLY))
2030     && !changesPending
2031     && (loopEnd == -1)
2032     && EOM)) {
2033
2034     long oldCheckPointMillis = checkPointMillis;
2035     long loopEndTick = loopEnd;
2036     if (loopEndTick == -1) {
2037         loopEndTick = lastTick;
2038     }
2039
2040     // need to loop back!
2041     if (loopCount != LOOP_CONTINUOUSLY) {
2042         currLoopCounter--;
2043     }
2044     if (DEBUG_PUMP) Printer.println("Execute_loop:_lastTick="+lastTick
2045                                     + "_loopEnd="+loopEnd
2046                                     + "_jumping_to_loopStart="+loopStart
2047                                     + "_new_currLoopCounter="+currLoopCounter)
2048                                     ;
2049     setTickPos(loopStart);
2050     // now patch the checkPointMillis so that
2051     // it points to the exact beginning of when the loop was finished
2052
2053     // $$$fb TODO: although this is mathematically correct (i.e. the loop position
2054     //              is correct, and doesn't drift away with several repetition,
2055     //              there is a slight lag when looping back, probably caused
2056     //              by the chasing.
2057
2058     checkPointMillis = oldCheckPointMillis + tick2millis(loopEndTick -
2059                                                         checkPointTick);
2060     checkPointTick = loopStart;
2061     if (DEBUG_PUMP) Printer.println("_Setting_currMillis="+currMillis
2062                                     + "_new_checkPointMillis="+
2063                                     checkPointMillis
2064                                     + "_new_checkPointTick="+checkPointTick);
2065     // no need for reindexing, is done in setTickPos
2066     needReindex = false;
2067     changesPending = false;
2068     // reset doLoop flag

```

```
2066         doLoop = false;
2067         EOM = false;
2068     }
2069 } while (changesPending);
2070
2071     return EOM;
2072 }
2073
2074 } // class DataPump
2075
2076 }
```

65 com/sun/media/sound/SF2GlobalRegion.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 /**
28 * Soundfont global region.
29 *
30 * @author Karl Helgason
31 */
32 public class SF2GlobalRegion extends SF2Region {
33 }
```

66 com/sun/media/sound/SF2Instrument.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.ArrayList;
28 import java.util.HashMap;
29 import java.util.List;
30 import java.util.Map;
31
32 import javax.sound.midi.Patch;
33
34 /**
35  * Soundfont instrument.
36  *
37  * @author Karl Helgason
38  */
39 public class SF2Instrument extends ModelInstrument {
40
41     protected String name = "";
42     protected int preset = 0;
43     protected int bank = 0;
44     protected long library = 0;
45     protected long genre = 0;
46     protected long morphology = 0;
47     protected SF2GlobalRegion globalregion = null;
48     protected List<SF2InstrumentRegion> regions
49         = new ArrayList<SF2InstrumentRegion>();
50
51     public SF2Instrument() {
52         super(null, null, null, null);
53     }
54
55     public SF2Instrument(SF2Soundbank soundbank) {
56         super(soundbank, null, null, null);
57     }
58
59     public String getName() {
60         return name;
```

```

61     }
62
63     public void setName(String name) {
64         this.name = name;
65     }
66
67     public Patch getPatch() {
68         if (bank == 128)
69             return new ModelPatch(0, preset, true);
70         else
71             return new ModelPatch(bank << 7, preset, false);
72     }
73
74     public void setPatch(Patch patch) {
75         if (patch instanceof ModelPatch && ((ModelPatch) patch).isPercussion()) {
76             bank = 128;
77             preset = patch.getProgram();
78         } else {
79             bank = patch.getBank() >> 7;
80             preset = patch.getProgram();
81         }
82     }
83
84     public Object getData() {
85         return null;
86     }
87
88     public long getGenre() {
89         return genre;
90     }
91
92     public void setGenre(long genre) {
93         this.genre = genre;
94     }
95
96     public long getLibrary() {
97         return library;
98     }
99
100    public void setLibrary(long library) {
101        this.library = library;
102    }
103
104    public long getMorphology() {
105        return morphology;
106    }
107
108    public void setMorphology(long morphology) {
109        this.morphology = morphology;
110    }
111
112    public List<SF2InstrumentRegion> getRegions() {
113        return regions;
114    }
115
116    public SF2GlobalRegion getGlobalRegion() {
117        return globalregion;
118    }
119
120    public void setGlobalZone(SF2GlobalRegion zone) {
121        globalregion = zone;
122    }

```



```

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];

```

```

185     }
186     if (presetzone.contains(SF2Region.GENERATOR_KEYRANGE)) {
187         byte[] bytes = presetzone.getBytes(
188             SF2Region.GENERATOR_KEYRANGE);
189         if (bytes[0] > keyfrom)
190             keyfrom = bytes[0];
191         if (bytes[1] < keyto)
192             keyto = bytes[1];
193     }
194     if (presetzone.contains(SF2Region.GENERATOR_VELRANGE)) {
195         byte[] bytes = presetzone.getBytes(
196             SF2Region.GENERATOR_VELRANGE);
197         if (bytes[0] > velfrom)
198             velfrom = bytes[0];
199         if (bytes[1] < velto)
200             velto = bytes[1];
201     }
202     performer.setKeyFrom(keyfrom);
203     performer.setKeyTo(keyto);
204     performer.setVelFrom(velfrom);
205     performer.setVelTo(velto);
206
207     int startAddrsOffset = layerzone.getShort(
208         SF2Region.GENERATOR_STARTADDRSOFFSET);
209     int endAddrsOffset = layerzone.getShort(
210         SF2Region.GENERATOR_ENDADDRSOFFSET);
211     int startloopAddrsOffset = layerzone.getShort(
212         SF2Region.GENERATOR_STARTLOOPADDRSOFFSET);
213     int endloopAddrsOffset = layerzone.getShort(
214         SF2Region.GENERATOR_ENDLOOPADDRSOFFSET);
215
216     startAddrsOffset += layerzone.getShort(
217         SF2Region.GENERATOR_STARTADDRSCOARSEOFFSET) * 32768;
218     endAddrsOffset += layerzone.getShort(
219         SF2Region.GENERATOR_ENDADDRSCOARSEOFFSET) * 32768;
220     startloopAddrsOffset += layerzone.getShort(
221         SF2Region.GENERATOR_STARTLOOPADDRSCOARSEOFFSET) * 32768;
222     endloopAddrsOffset += layerzone.getShort(
223         SF2Region.GENERATOR_ENDLOOPADDRSCOARSEOFFSET) * 32768;
224     startloopAddrsOffset -= startAddrsOffset;
225     endloopAddrsOffset -= startAddrsOffset;
226
227     SF2Sample sample = layerzone.getSample();
228     int rootkey = sample.originalPitch;
229     if (layerzone.getShort(SF2Region.GENERATOR_OVERRIDINGROOTKEY) != -1) {
230         rootkey = layerzone.getShort(
231             SF2Region.GENERATOR_OVERRIDINGROOTKEY);
232     }
233     float pitchcorrection = (-rootkey * 100) + sample.pitchCorrection;
234     ModelByteBuffer buff = sample.getDataBuffer();
235     ModelByteBuffer buff24 = sample.getData24Buffer();
236
237     if (startAddrsOffset != 0 || endAddrsOffset != 0) {
238         buff = buff.subbuffer(startAddrsOffset * 2,
239             buff.capacity() + endAddrsOffset * 2);
240         if (buff24 != null) {
241             buff24 = buff24.subbuffer(startAddrsOffset,
242                 buff24.capacity() + endAddrsOffset);
243         }
244
245         /*
246         if (startAddrsOffset < 0)

```

```

247         startAddrsOffset = 0;
248         if (endAddrsOffset > (buff.capacity()/2-startAddrsOffset))
249             startAddrsOffset = (int)buff.capacity()/2-startAddrsOffset;
250         byte[] data = buff.array();
251         int off = (int)buff.arrayOffset() + startAddrsOffset*2;
252         int len = (int)buff.capacity() + endAddrsOffset*2;
253         if (off+len > data.length)
254             len = data.length - off;
255         buff = new ModelByteBuffer(data, off, len);
256         if(buff24 != null) {
257             data = buff.array();
258             off = (int)buff.arrayOffset() + startAddrsOffset;
259             len = (int)buff.capacity() + endAddrsOffset;
260             buff24 = new ModelByteBuffer(data, off, len);
261         }
262         */
263     }
264
265     ModelByteBufferWavetable osc = new ModelByteBufferWavetable(
266         buff, sample.getFormat(), pitchcorrection);
267     if (buff24 != null)
268         osc.set8BitExtensionBuffer(buff24);
269
270     Map<Integer, Short> generators = new HashMap<Integer, Short>();
271     if (layerglobal != null)
272         generators.putAll(layerglobal.getGenerators());
273     generators.putAll(layerzone.getGenerators());
274     for (Map.Entry<Integer, Short> gen : pgenerators.entrySet()) {
275         short val;
276         if (!generators.containsKey(gen.getKey()))
277             val = layerzone.getShort(gen.getKey());
278         else
279             val = generators.get(gen.getKey());
280         val += gen.getValue();
281         generators.put(gen.getKey(), val);
282     }
283
284     // SampleMode:
285     // 0 indicates a sound reproduced with no loop
286     // 1 indicates a sound which loops continuously
287     // 2 is unused but should be interpreted as indicating no loop
288     // 3 indicates a sound which loops for the duration of key
289     // depression then proceeds to play the remainder of the sample.
290     int sampleMode = getGeneratorValue(generators,
291         SF2Region.GENERATOR_SAMPLEMODES);
292     if ((sampleMode == 1) || (sampleMode == 3)) {
293         if (sample.startLoop >= 0 && sample.endLoop > 0) {
294             osc.setLoopStart((int)(sample.startLoop
295                 + startloopAddrsOffset));
296             osc.setLoopLength((int)(sample.endLoop - sample.startLoop
297                 + endloopAddrsOffset - startloopAddrsOffset));
298             if (sampleMode == 1)
299                 osc.setLoopType(ModelWavetable.LOOP_TYPE_FORWARD);
300             if (sampleMode == 3)
301                 osc.setLoopType(ModelWavetable.LOOP_TYPE_RELEASE);
302         }
303     }
304     performer.getOscillators().add(osc);
305
306     short volDelay = getGeneratorValue(generators,
307         SF2Region.GENERATOR_DELAYVOLENV);
308

```

```

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,

```

```

371         SF2Region.GENERATOR_ATTACKMODENV);
372     short modHold = getGeneratorValue(generators,
373         SF2Region.GENERATOR_HOLDMODENV);
374     short modDecay = getGeneratorValue(generators,
375         SF2Region.GENERATOR_DECAYMODENV);
376     short modSustain = getGeneratorValue(generators,
377         SF2Region.GENERATOR_SUSTAINMODENV);
378     short modRelease = getGeneratorValue(generators,
379         SF2Region.GENERATOR_RELEASEMODENV);
380
381
382     if (modHold != -12000) {
383         short modKeyNumToHold = getGeneratorValue(generators,
384             SF2Region.GENERATOR_KEYNUMTOMODENVHOLD);
385         modHold += 60 * modKeyNumToHold;
386         float fvalue = -modKeyNumToHold * 128;
387         ModelIdentifier src = ModelSource.SOURCE_NOTEON_KEYNUMBER;
388         ModelIdentifier dest = ModelDestination.DESTINATION_EG2_HOLD;
389         performer.getConnectionBlocks().add(
390             new ModelConnectionBlock(new ModelSource(src),
391                 fvalue, new ModelDestination(dest)));
392     }
393     if (modDecay != -12000) {
394         short modKeyNumToDecay = getGeneratorValue(generators,
395             SF2Region.GENERATOR_KEYNUMTOMODENVDECAY);
396         modDecay += 60 * modKeyNumToDecay;
397         float fvalue = -modKeyNumToDecay * 128;
398         ModelIdentifier src = ModelSource.SOURCE_NOTEON_KEYNUMBER;
399         ModelIdentifier dest = ModelDestination.DESTINATION_EG2_DECAY;
400         performer.getConnectionBlocks().add(
401             new ModelConnectionBlock(new ModelSource(src),
402                 fvalue, new ModelDestination(dest)));
403     }
404
405     addTimecentValue(performer,
406         ModelDestination.DESTINATION_EG2_DELAY, modDelay);
407     addTimecentValue(performer,
408         ModelDestination.DESTINATION_EG2_ATTACK, modAttack);
409     addTimecentValue(performer,
410         ModelDestination.DESTINATION_EG2_HOLD, modHold);
411     addTimecentValue(performer,
412         ModelDestination.DESTINATION_EG2_DECAY, modDecay);
413     if (modSustain < 0)
414         modSustain = 0;
415     if (modSustain > 1000)
416         modSustain = 1000;
417     addValue(performer, ModelDestination.DESTINATION_EG2_SUSTAIN,
418         1000 - modSustain);
419     addTimecentValue(performer,
420         ModelDestination.DESTINATION_EG2_RELEASE, modRelease);
421
422     if (getGeneratorValue(generators,
423         SF2Region.GENERATOR_MODENVTOFILTERFC) != 0) {
424         double fvalue = getGeneratorValue(generators,
425             SF2Region.GENERATOR_MODENVTOFILTERFC);
426         ModelIdentifier src = ModelSource.SOURCE_EG2;
427         ModelIdentifier dest
428             = ModelDestination.DESTINATION_FILTER_FREQ;
429         performer.getConnectionBlocks().add(
430             new ModelConnectionBlock(new ModelSource(src),
431                 fvalue, new ModelDestination(dest)));
432     }

```

```

433         if (getGeneratorValue(generators,
434             SF2Region.GENERATOR_MODENVTOPITCH) != 0) {
435             double fvalue = getGeneratorValue(generators,
436                 SF2Region.GENERATOR_MODENVTOPITCH);
437             ModelIdentifier src = ModelSource.SOURCE_EG2;
438             ModelIdentifier dest = ModelDestination.DESTINATION_PITCH;
439             performer.getConnectionBlocks().add(
440                 new ModelConnectionBlock(new ModelSource(src),
441                     fvalue, new ModelDestination(dest)));
442         }
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_LF01_DELAY, lfo_delay);
459         addValue(performer,
460             ModelDestination.DESTINATION_LF01_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_LF02_DELAY, vib_delay);
469     addValue(performer,
470         ModelDestination.DESTINATION_LF02_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_LF02;
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_LF01;
492         ModelIdentifier dest = ModelDestination.DESTINATION_FILTER_FREQ;
493         performer.getConnectionBlocks().add(
494             new ModelConnectionBlock(

```

```

495         new ModelSource(src,
496             ModelStandardTransform.DIRECTION_MIN2MAX,
497             ModelStandardTransform.POLARITY_BIPOLAR),
498         fvalue, new ModelDestination(dest));
499     }
500
501     if (getGeneratorValue(generators,
502         SF2Region.GENERATOR_MODLFOTOPITCH) != 0) {
503         double fvalue = getGeneratorValue(generators,
504             SF2Region.GENERATOR_MODLFOTOPITCH);
505         ModelIdentifier src = ModelSource.SOURCE_LF01;
506         ModelIdentifier dest = ModelDestination.DESTINATION_PITCH;
507         performer.getConnectionBlocks().add(
508             new ModelConnectionBlock(
509                 new ModelSource(src,
510                     ModelStandardTransform.DIRECTION_MIN2MAX,
511                     ModelStandardTransform.POLARITY_BIPOLAR),
512                 fvalue, new ModelDestination(dest)));
513     }
514
515     if (getGeneratorValue(generators,
516         SF2Region.GENERATOR_MODLFOTOVOLUME) != 0) {
517         double fvalue = getGeneratorValue(generators,
518             SF2Region.GENERATOR_MODLFOTOVOLUME);
519         ModelIdentifier src = ModelSource.SOURCE_LF01;
520         ModelIdentifier dest = ModelDestination.DESTINATION_GAIN;
521         performer.getConnectionBlocks().add(
522             new ModelConnectionBlock(
523                 new ModelSource(src,
524                     ModelStandardTransform.DIRECTION_MIN2MAX,
525                     ModelStandardTransform.POLARITY_BIPOLAR),
526                 fvalue, new ModelDestination(dest)));
527     }
528
529     if (layerzone.getShort(SF2Region.GENERATOR_KEYNUM) != -1) {
530         double val = layerzone.getShort(SF2Region.GENERATOR_KEYNUM)/128.0;
531         addValue(performer, ModelDestination.DESTINATION_KEYNUMBER, val);
532     }
533
534     if (layerzone.getShort(SF2Region.GENERATOR_VELOCITY) != -1) {
535         double val = layerzone.getShort(SF2Region.GENERATOR_VELOCITY)
536             / 128.0;
537         addValue(performer, ModelDestination.DESTINATION_VELOCITY, val);
538     }
539
540     if (getGeneratorValue(generators,
541         SF2Region.GENERATOR_INITIALFILTERFC) < 13500) {
542         short filter_freq = getGeneratorValue(generators,
543             SF2Region.GENERATOR_INITIALFILTERFC);
544         short filter_q = getGeneratorValue(generators,
545             SF2Region.GENERATOR_INITIALFILTERQ);
546         addValue(performer,
547             ModelDestination.DESTINATION_FILTER_FREQ, filter_freq);
548         addValue(performer,
549             ModelDestination.DESTINATION_FILTER_Q, filter_q);
550     }
551
552     int tune = 100 * getGeneratorValue(generators,
553         SF2Region.GENERATOR_COARSETUNE);
554     tune += getGeneratorValue(generators,
555         SF2Region.GENERATOR_FINETUNE);
556     if (tune != 0) {

```

```

557         addValue(performer,
558                 ModelDestination.DESTINATION_PITCH, (short) tune);
559     }
560     if (getGeneratorValue(generators, SF2Region.GENERATOR_PAN) != 0) {
561         short val = getGeneratorValue(generators,
562                 SF2Region.GENERATOR_PAN);
563         addValue(performer, ModelDestination.DESTINATION_PAN, val);
564     }
565     if (getGeneratorValue(generators, SF2Region.GENERATOR_INITIALATTENUATION) != 0) {
566         short val = getGeneratorValue(generators,
567                 SF2Region.GENERATOR_INITIALATTENUATION);
568         addValue(performer,
569                 ModelDestination.DESTINATION_GAIN, -0.376287f * val);
570     }
571     if (getGeneratorValue(generators,
572                 SF2Region.GENERATOR_CHORUSEFFECTSEND) != 0) {
573         short val = getGeneratorValue(generators,
574                 SF2Region.GENERATOR_CHORUSEFFECTSEND);
575         addValue(performer, ModelDestination.DESTINATION_CHORUS, val);
576     }
577     if (getGeneratorValue(generators,
578                 SF2Region.GENERATOR_REVERBEFFECTSEND) != 0) {
579         short val = getGeneratorValue(generators,
580                 SF2Region.GENERATOR_REVERBEFFECTSEND);
581         addValue(performer, ModelDestination.DESTINATION_REVERB, val);
582     }
583     if (getGeneratorValue(generators,
584                 SF2Region.GENERATOR_SCALETUNING) != 100) {
585         short fvalue = getGeneratorValue(generators,
586                 SF2Region.GENERATOR_SCALETUNING);
587         if (fvalue == 0) {
588             ModelIdentifier dest = ModelDestination.DESTINATION_PITCH;
589             performer.getConnectionBlocks().add(
590                 new ModelConnectionBlock(null, rootkey * 100,
591                     new ModelDestination(dest)));
592         } else {
593             ModelIdentifier dest = ModelDestination.DESTINATION_PITCH;
594             performer.getConnectionBlocks().add(
595                 new ModelConnectionBlock(null, rootkey * (100 - fvalue),
596                     new ModelDestination(dest)));
597         }
598
599         ModelIdentifier src = ModelSource.SOURCE_NOTEON_KEYNUMBER;
600         ModelIdentifier dest = ModelDestination.DESTINATION_PITCH;
601         performer.getConnectionBlocks().add(
602             new ModelConnectionBlock(new ModelSource(src),
603                 128 * fvalue, new ModelDestination(dest)));
604     }
605 }
606
607 performer.getConnectionBlocks().add(
608     new ModelConnectionBlock(
609         new ModelSource(ModelSource.SOURCE_NOTEON_VELOCITY,
610             new ModelTransform() {
611                 public double transform(double value) {
612                     if (value < 0.5)
613                         return 1 - value * 2;
614                     else
615                         return 0;
616                 }
617             )),
618     -2400,

```



```

619         new ModelDestination(
620             ModelDestination.DESTINATION_FILTER_FREQ));
621
622
623     performer.getConnectionBlocks().add(
624         new ModelConnectionBlock(
625             new ModelSource(ModelSource.SOURCE_LFO2,
626                 ModelStandardTransform.DIRECTION_MIN2MAX,
627                 ModelStandardTransform.POLARITY_BIPOLAR,
628                 ModelStandardTransform.TRANSFORM_LINEAR),
629             new ModelSource(new ModelIdentifier("midi_cc", "1", 0),
630                 ModelStandardTransform.DIRECTION_MIN2MAX,
631                 ModelStandardTransform.POLARITY_UNIPOLAR,
632                 ModelStandardTransform.TRANSFORM_LINEAR),
633             50, new ModelDestination(
634                 ModelDestination.DESTINATION_PITCH));
635
636     if (layer.getGlobalRegion() != null) {
637         for (SF2Modulator modulator
638             : layer.getGlobalRegion().getModulators()) {
639             convertModulator(performer, modulator);
640         }
641     }
642     for (SF2Modulator modulator : layerzone.getModulators())
643         convertModulator(performer, modulator);
644
645     if (presetglobal != null) {
646         for (SF2Modulator modulator : presetglobal.getModulators())
647             convertModulator(performer, modulator);
648     }
649     for (SF2Modulator modulator : presetzone.getModulators())
650         convertModulator(performer, modulator);
651
652     }
653 }
654 return performers;
655 }
656
657 private void convertModulator(ModelPerformer performer,
658     SF2Modulator modulator) {
659     ModelSource src1 = convertSource(modulator.getSourceOperator());
660     ModelSource src2 = convertSource(modulator.getAmountSourceOperator());
661     if (src1 == null && modulator.getSourceOperator() != 0)
662         return;
663     if (src2 == null && modulator.getAmountSourceOperator() != 0)
664         return;
665     double amount = modulator.getAmount();
666     double[] amountcorrection = new double[1];
667     ModelSource[] extrasrc = new ModelSource[1];
668     amountcorrection[0] = 1;
669     ModelDestination dst = convertDestination(
670         modulator.getDestinationOperator(), amountcorrection, extrasrc);
671     amount *= amountcorrection[0];
672     if (dst == null)
673         return;
674     if (modulator.getTransportOperator() == SF2Modulator.TRANSFORM_ABSOLUTE) {
675         ((ModelStandardTransform)dst.getTransform()).setTransform(
676             ModelStandardTransform.TRANSFORM_ABSOLUTE);
677     }
678     ModelConnectionBlock conn = new ModelConnectionBlock(src1, src2, amount, dst);
679     if (extrasrc[0] != null)
680         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;

```

```

743     case SF2Region.GENERATOR_CHORUSEFFECTSSEND:
744         id = ModelDestination.DESTINATION_CHORUS;
745         break;
746     case SF2Region.GENERATOR_REVERBEFFECTSSEND:
747         id = ModelDestination.DESTINATION_REVERB;
748         break;
749     case SF2Region.GENERATOR_PAN:
750         id = ModelDestination.DESTINATION_PAN;
751         break;
752     case SF2Region.GENERATOR_DELAYMODLFO:
753         id = ModelDestination.DESTINATION_LF01_DELAY;
754         break;
755     case SF2Region.GENERATOR_FREQMODLFO:
756         id = ModelDestination.DESTINATION_LF01_FREQ;
757         break;
758     case SF2Region.GENERATOR_DELAYVIBLFO:
759         id = ModelDestination.DESTINATION_LF02_DELAY;
760         break;
761     case SF2Region.GENERATOR_FREQVIBLFO:
762         id = ModelDestination.DESTINATION_LF02_FREQ;
763         break;
764
765     case SF2Region.GENERATOR_DELAYMODENV:
766         id = ModelDestination.DESTINATION_EG2_DELAY;
767         break;
768     case SF2Region.GENERATOR_ATTACKMODENV:
769         id = ModelDestination.DESTINATION_EG2_ATTACK;
770         break;
771     case SF2Region.GENERATOR_HOLDMODENV:
772         id = ModelDestination.DESTINATION_EG2_HOLD;
773         break;
774     case SF2Region.GENERATOR_DECAYMODENV:
775         id = ModelDestination.DESTINATION_EG2_DECAY;
776         break;
777     case SF2Region.GENERATOR_SUSTAINMODENV:
778         id = ModelDestination.DESTINATION_EG2_SUSTAIN;
779         amountcorrection[0] = -1;
780         break;
781     case SF2Region.GENERATOR_RELEASEMODENV:
782         id = ModelDestination.DESTINATION_EG2_RELEASE;
783         break;
784     case SF2Region.GENERATOR_DELAYVOLENV:
785         id = ModelDestination.DESTINATION_EG1_DELAY;
786         break;
787     case SF2Region.GENERATOR_ATTACKVOLENV:
788         id = ModelDestination.DESTINATION_EG1_ATTACK;
789         break;
790     case SF2Region.GENERATOR_HOLDVOLENV:
791         id = ModelDestination.DESTINATION_EG1_HOLD;
792         break;
793     case SF2Region.GENERATOR_DECAYVOLENV:
794         id = ModelDestination.DESTINATION_EG1_DECAY;
795         break;
796     case SF2Region.GENERATOR_SUSTAINVOLENV:
797         id = ModelDestination.DESTINATION_EG1_SUSTAIN;
798         amountcorrection[0] = -1;
799         break;
800     case SF2Region.GENERATOR_RELEASEVOLENV:
801         id = ModelDestination.DESTINATION_EG1_RELEASE;
802         break;
803     case SF2Region.GENERATOR_KEYNUM:
804         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_LF02,
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_LF01,
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_LF01,
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_LF01,
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;

```

```

867         extrasrc[0] = new ModelSource(
868             ModelSource.SOURCE_EG2,
869             ModelStandardTransform.DIRECTION_MIN2MAX,
870             ModelStandardTransform.POLARITY_BIPOLAR);
871         break;
872
873     default:
874         break;
875 }
876 if (id != null)
877     return new ModelDestination(id);
878 return null;
879 }
880
881 private void addTimecentValue(ModelPerformer performer,
882     ModelIdentifier dest, short value) {
883     double fvalue;
884     if (value == -12000)
885         fvalue = Double.NEGATIVE_INFINITY;
886     else
887         fvalue = value;
888     performer.getConnectionBlocks().add(
889         new ModelConnectionBlock(fvalue, new ModelDestination(dest)));
890 }
891
892 private void addValue(ModelPerformer performer,
893     ModelIdentifier dest, short value) {
894     double fvalue = value;
895     performer.getConnectionBlocks().add(
896         new ModelConnectionBlock(fvalue, new ModelDestination(dest)));
897 }
898
899 private void addValue(ModelPerformer performer,
900     ModelIdentifier dest, double value) {
901     double fvalue = value;
902     performer.getConnectionBlocks().add(
903         new ModelConnectionBlock(fvalue, new ModelDestination(dest)));
904 }
905
906 private short getGeneratorValue(Map<Integer, Short> generators, int gen) {
907     if (generators.containsKey(gen))
908         return generators.get(gen);
909     return SF2Region.getDefaultValue(gen);
910 }
911 }

```

67 com/sun/media/sound/SF2InstrumentRegion.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 /**
28  * Soundfont instrument region.
29  *
30  * @author Karl Helgason
31  */
32 public class SF2InstrumentRegion extends SF2Region {
33
34     protected SF2Layer layer;
35
36     public SF2Layer getLayer() {
37         return layer;
38     }
39
40     public void setLayer(SF2Layer layer) {
41         this.layer = layer;
42     }
43 }
```

68 com/sun/media/sound/SF2Layer.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.ArrayList;
28 import java.util.List;
29
30 import javax.sound.midi.SoundbankResource;
31
32 /**
33  * Soundfont layer.
34  *
35  * @author Karl Helgason
36  */
37 public class SF2Layer extends SoundbankResource {
38
39     protected String name = "";
40     protected SF2GlobalRegion globalregion = null;
41     protected List<SF2LayerRegion> regions = new ArrayList<SF2LayerRegion>();
42
43     public SF2Layer(SF2Soundbank soundBank) {
44         super(soundBank, null, null);
45     }
46
47     public SF2Layer() {
48         super(null, null, null);
49     }
50
51     public Object getData() {
52         return null;
53     }
54
55     public String getName() {
56         return name;
57     }
58
59     public void setName(String name) {
60         this.name = name;
```

```
61     }
62
63     public List<SF2LayerRegion> getRegions() {
64         return regions;
65     }
66
67     public SF2GlobalRegion getGlobalRegion() {
68         return globalregion;
69     }
70
71     public void setGlobalZone(SF2GlobalRegion zone) {
72         globalregion = zone;
73     }
74
75     public String toString() {
76         return "Layer:␣" + name;
77     }
78 }
```


69 com/sun/media/sound/SF2LayerRegion.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 /**
28  * Soundfont layer region.
29  *
30  * @author Karl Helgason
31  */
32 public class SF2LayerRegion extends SF2Region {
33
34     protected SF2Sample sample;
35
36     public SF2Sample getSample() {
37         return sample;
38     }
39
40     public void setSample(SF2Sample sample) {
41         this.sample = sample;
42     }
43 }
```

70 com/sun/media/sound/SF2Modulator.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 /**
28 * Soundfont modulator container.
29 *
30 * @author Karl Helgason
31 */
32 public class SF2Modulator {
33
34     public final static int SOURCE_NONE = 0;
35     public final static int SOURCE_NOTE_ON_VELOCITY = 2;
36     public final static int SOURCE_NOTE_ON_KEYNUMBER = 3;
37     public final static int SOURCE_POLY_PRESSURE = 10;
38     public final static int SOURCE_CHANNEL_PRESSURE = 13;
39     public final static int SOURCE_PITCH_WHEEL = 14;
40     public final static int SOURCE_PITCH_SENSITIVITY = 16;
41     public final static int SOURCE_MIDI_CONTROL = 128 * 1;
42     public final static int SOURCE_DIRECTION_MIN_MAX = 256 * 0;
43     public final static int SOURCE_DIRECTION_MAX_MIN = 256 * 1;
44     public final static int SOURCE_POLARITY_UNIPOLAR = 512 * 0;
45     public final static int SOURCE_POLARITY_BIPOLAR = 512 * 1;
46     public final static int SOURCE_TYPE_LINEAR = 1024 * 0;
47     public final static int SOURCE_TYPE_CONCAVE = 1024 * 1;
48     public final static int SOURCE_TYPE_CONVEX = 1024 * 2;
49     public final static int SOURCE_TYPE_SWITCH = 1024 * 3;
50     public final static int TRANSFORM_LINEAR = 0;
51     public final static int TRANSFORM_ABSOLUTE = 2;
52     protected int sourceOperator;
53     protected int destinationOperator;
54     protected short amount;
55     protected int amountSourceOperator;
56     protected int transportOperator;
57
58     public short getAmount() {
59         return amount;
60     }
```

```

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

```
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.ArrayList;
28 import java.util.HashMap;
29 import java.util.List;
30 import java.util.Map;
31
32 /**
33  * Soundfont general region.
34  *
35  * @author Karl Helgason
36  */
37 public class SF2Region {
38
39     public final static int GENERATOR_STARTADDRSOFFSET = 0;
40     public final static int GENERATOR_ENDADDRSOFFSET = 1;
41     public final static int GENERATOR_STARTLOOPADDRSOFFSET = 2;
42     public final static int GENERATOR_ENDLOOPADDRSOFFSET = 3;
43     public final static int GENERATOR_STARTADDRSCOARSEOFFSET = 4;
44     public final static int GENERATOR_MODLFOTOPITCH = 5;
45     public final static int GENERATOR_VIBLFOTOPITCH = 6;
46     public final static int GENERATOR_MODENVTOPITCH = 7;
47     public final static int GENERATOR_INITIALFILTERFC = 8;
48     public final static int GENERATOR_INITIALFILTERQ = 9;
49     public final static int GENERATOR_MODLFOTOFILTERFC = 10;
50     public final static int GENERATOR_MODENVTOFILTERFC = 11;
51     public final static int GENERATOR_ENDADDRSCOARSEOFFSET = 12;
52     public final static int GENERATOR_MODLFOTOVOLUME = 13;
53     public final static int GENERATOR_UNUSED1 = 14;
54     public final static int GENERATOR_CHORUSEFFECTSEND = 15;
55     public final static int GENERATOR_REVERBEFFECTSEND = 16;
56     public final static int GENERATOR_PAN = 17;
57     public final static int GENERATOR_UNUSED2 = 18;
58     public final static int GENERATOR_UNUSED3 = 19;
59     public final static int GENERATOR_UNUSED4 = 20;
60     public final static int GENERATOR_DELAYMODLFO = 21;
```

```

61 public final static int GENERATOR_FREQMODLFO = 22;
62 public final static int GENERATOR_DELAYVIBLFO = 23;
63 public final static int GENERATOR_FREQVIBLFO = 24;
64 public final static int GENERATOR_DELAYMODENV = 25;
65 public final static int GENERATOR_ATTACKMODENV = 26;
66 public final static int GENERATOR_HOLDMODENV = 27;
67 public final static int GENERATOR_DECAYMODENV = 28;
68 public final static int GENERATOR_SUSTAINMODENV = 29;
69 public final static int GENERATOR_RELEASEMODENV = 30;
70 public final static int GENERATOR_KEYNUMTOMODENVHOLD = 31;
71 public final static int GENERATOR_KEYNUMTOMODENVDECAY = 32;
72 public final static int GENERATOR_DELAYVOLENV = 33;
73 public final static int GENERATOR_ATTACKVOLENV = 34;
74 public final static int GENERATOR_HOLDVOLENV = 35;
75 public final static int GENERATOR_DECAYVOLENV = 36;
76 public final static int GENERATOR_SUSTAINVOLENV = 37;
77 public final static int GENERATOR_RELEASEVOLENV = 38;
78 public final static int GENERATOR_KEYNUMTOVOLENVHOLD = 39;
79 public final static int GENERATOR_KEYNUMTOVOLENVDECAY = 40;
80 public final static int GENERATOR_INSTRUMENT = 41;
81 public final static int GENERATOR_RESERVED1 = 42;
82 public final static int GENERATOR_KEYRANGE = 43;
83 public final static int GENERATOR_VELRANGE = 44;
84 public final static int GENERATOR_STARTLOOPADDRSCARSEOFFSET = 45;
85 public final static int GENERATOR_KEYNUM = 46;
86 public final static int GENERATOR_VELOCITY = 47;
87 public final static int GENERATOR_INITIALATTENUATION = 48;
88 public final static int GENERATOR_RESERVED2 = 49;
89 public final static int GENERATOR_ENDLOOPADDRSCARSEOFFSET = 50;
90 public final static int GENERATOR_COARSETUNE = 51;
91 public final static int GENERATOR_FINETUNE = 52;
92 public final static int GENERATOR_SAMPLEID = 53;
93 public final static int GENERATOR_SAMPLEMODES = 54;
94 public final static int GENERATOR_RESERVED3 = 55;
95 public final static int GENERATOR_SCALETUNING = 56;
96 public final static int GENERATOR_EXCLUSIVECLASS = 57;
97 public final static int GENERATOR_OVERRIDINGROOTKEY = 58;
98 public final static int GENERATOR_UNUSED5 = 59;
99 public final static int GENERATOR_ENDOPR = 60;
100 protected Map<Integer, Short> generators = new HashMap<Integer, Short>();
101 protected List<SF2Modulator> modulators = new ArrayList<SF2Modulator>();
102
103 public Map<Integer, Short> getGenerators() {
104     return generators;
105 }
106
107 public boolean contains(int generator) {
108     return generators.containsKey(generator);
109 }
110
111 static public short getDefaultValue(int generator) {
112     if (generator == 8) return (short)13500;
113     if (generator == 21) return (short)-12000;
114     if (generator == 23) return (short)-12000;
115     if (generator == 25) return (short)-12000;
116     if (generator == 26) return (short)-12000;
117     if (generator == 27) return (short)-12000;
118     if (generator == 28) return (short)-12000;
119     if (generator == 30) return (short)-12000;
120     if (generator == 33) return (short)-12000;
121     if (generator == 34) return (short)-12000;
122     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

```
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.io.InputStream;
28
29 import javax.sound.midi.Soundbank;
30 import javax.sound.midi.SoundbankResource;
31 import javax.sound.sampled.AudioFormat;
32 import javax.sound.sampled.AudioInputStream;
33
34 /**
35  * Soundfont sample storage.
36  *
37  * @author Karl Helgason
38  */
39 public class SF2Sample extends SoundbankResource {
40
41     protected String name = "";
42     protected long startLoop = 0;
43     protected long endLoop = 0;
44     protected long sampleRate = 44100;
45     protected int originalPitch = 60;
46     protected byte pitchCorrection = 0;
47     protected int sampleLink = 0;
48     protected int sampleType = 0;
49     protected ModelByteBuffer data;
50     protected ModelByteBuffer data24;
51
52     public SF2Sample(Soundbank soundBank) {
53         super(soundBank, null, AudioInputStream.class);
54     }
55
56     public SF2Sample() {
57         super(null, null, AudioInputStream.class);
58     }
59
60     public Object getData() {
```

```

61     AudioFormat format = getFormat();
62     /*
63     if (sampleFile != null) {
64         FileInputStream fis;
65         try {
66             fis = new FileInputStream(sampleFile);
67             RIFFReader riff = new RIFFReader(fis);
68             if (!riff.getFormat().equals("RIFF")) {
69                 throw new RIFFInvalidDataException(
70                     "Input stream is not a valid RIFF stream!");
71             }
72             if (!riff.getType().equals("sfbk")) {
73                 throw new RIFFInvalidDataException(
74                     "Input stream is not a valid SoundFont!");
75             }
76             while (riff.hasNextChunk()) {
77                 RIFFReader chunk = riff.nextChunk();
78                 if (chunk.getFormat().equals("LIST")) {
79                     if (chunk.getType().equals("sdta")) {
80                         while(chunk.hasNextChunk()) {
81                             RIFFReader chunkchunk = chunk.nextChunk();
82                             if(chunkchunk.getFormat().equals("smp1")) {
83                                 chunkchunk.skip(sampleOffset);
84                                 return new AudioInputStream(chunkchunk,
85                                     format, sampleLen);
86                             }
87                         }
88                     }
89                 }
90             }
91             return null;
92         } catch (Exception e) {
93             return new Throwable(e.toString());
94         }
95     }
96     */
97     InputStream is = data.getInputStream();
98     if (is == null)
99         return null;
100     return new AudioInputStream(is, format, data.capacity());
101 }
102
103 public ModelByteBuffer getDataBuffer() {
104     return data;
105 }
106
107 public ModelByteBuffer getData24Buffer() {
108     return data24;
109 }
110
111 public AudioFormat getFormat() {
112     return new AudioFormat(sampleRate, 16, 1, true, false);
113 }
114
115 public void setData(ModelByteBuffer data) {
116     this.data = data;
117 }
118
119 public void setData(byte[] data) {
120     this.data = new ModelByteBuffer(data);
121 }
122

```



```

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

```

```

185     public void setSampleLink(int sampleLink) {
186         this.sampleLink = sampleLink;
187     }
188
189     public long getSampleRate() {
190         return sampleRate;
191     }
192
193     public void setSampleRate(long sampleRate) {
194         this.sampleRate = sampleRate;
195     }
196
197     public int getSampleType() {
198         return sampleType;
199     }
200
201     public void setSampleType(int sampleType) {
202         this.sampleType = sampleType;
203     }
204
205     public long getStartLoop() {
206         return startLoop;
207     }
208
209     public void setStartLoop(long startLoop) {
210         this.startLoop = startLoop;
211     }
212
213     public String toString() {
214         return "Sample:␣" + name;
215     }
216 }

```

73 com/sun/media/sound/SF2Soundbank.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.io.File;
28 import java.io.FileInputStream;
29 import java.io.IOException;
30 import java.io.InputStream;
31 import java.io.OutputStream;
32 import java.net.URL;
33 import java.util.ArrayList;
34 import java.util.Arrays;
35 import java.util.Iterator;
36 import java.util.List;
37 import java.util.Map;
38
39 import javax.sound.midi.Instrument;
40 import javax.sound.midi.Patch;
41 import javax.sound.midi.Soundbank;
42 import javax.sound.midi.SoundbankResource;
43
44 /**
45 * A SoundFont 2.04 soundbank reader.
46 *
47 * Based on SoundFont 2.04 specification from:
48 * <p> http://developer.creative.com <br>
49 * http://www.soundfont.com/ ;
50 *
51 * @author Karl Helgason
52 */
53 public class SF2Soundbank implements Soundbank {
54
55     // version of the Sound Font RIFF file
56     protected int major = 2;
57     protected int minor = 1;
58     // target Sound Engine
59     protected String targetEngine = "EMU8000";
60     // Sound Font Bank Name
```

```

61     protected String name = "untitled";
62     // Sound ROM Name
63     protected String romName = null;
64     // Sound ROM Version
65     protected int romVersionMajor = -1;
66     protected int romVersionMinor = -1;
67     // Date of Creation of the Bank
68     protected String creationDate = null;
69     // Sound Designers and Engineers for the Bank
70     protected String engineers = null;
71     // Product for which the Bank was intended
72     protected String product = null;
73     // Copyright message
74     protected String copyright = null;
75     // Comments
76     protected String comments = null;
77     // The SoundFont tools used to create and alter the bank
78     protected String tools = null;
79     // The Sample Data loaded from the SoundFont
80     private ModelByteBuffer sampleData = null;
81     private ModelByteBuffer sampleData24 = null;
82     private File sampleFile = null;
83     private boolean largeFormat = false;
84     private List<SF2Instrument> instruments = new ArrayList<SF2Instrument>();
85     private List<SF2Layer> layers = new ArrayList<SF2Layer>();
86     private List<SF2Sample> samples = new ArrayList<SF2Sample>();
87
88     public SF2Soundbank() {
89     }
90
91     public SF2Soundbank(URL url) throws IOException {
92
93         InputStream is = url.openStream();
94         try {
95             readSoundbank(is);
96         } finally {
97             is.close();
98         }
99     }
100
101     public SF2Soundbank(File file) throws IOException {
102         largeFormat = true;
103         sampleFile = file;
104         InputStream is = new FileInputStream(file);
105         try {
106             readSoundbank(is);
107         } finally {
108             is.close();
109         }
110     }
111
112     public SF2Soundbank(InputStream inputstream) throws IOException {
113         readSoundbank(inputstream);
114     }
115
116     private void readSoundbank(InputStream inputstream) throws IOException {
117         RIFFReader riff = new RIFFReader(inputstream);
118         if (!riff.getFormat().equals("RIFF")) {
119             throw new RIFFInvalidFormatException(
120                 "Input_stream_is_not_a_valid_RIFF_stream!");
121         }
122         if (!riff.getType().equals("sfbk")) {

```

```

123         throw new RIFFInvalidFormatException(
124             "InputStream 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;

```

```

185         } else {
186             chunk.readFully(sampleData, read, avail - read);
187             read = avail;
188         }
189     }
190     }
191     this.sampleData = new ModelByteBuffer(sampleData);
192     //chunk.read(sampleData);
193 } else {
194     this.sampleData = new ModelByteBuffer(sampleFile,
195         chunk.getFilePointer(), chunk.available());
196 }
197 }
198 if (chunk.getFormat().equals("sm24")) {
199     if (!largeFormat) {
200         byte[] sampleData24 = new byte[chunk.available()];
201         //chunk.read(sampleData24);
202
203         int read = 0;
204         int avail = chunk.available();
205         while (read != avail) {
206             if (avail - read > 65536) {
207                 chunk.readFully(sampleData24, read, 65536);
208                 read += 65536;
209             } else {
210                 chunk.readFully(sampleData24, read, avail - read);
211                 read = avail;
212             }
213         }
214         this.sampleData24 = new ModelByteBuffer(sampleData24);
215     } else {
216         this.sampleData24 = new ModelByteBuffer(sampleFile,
217             chunk.getFilePointer(), chunk.available());
218     }
219 }
220 }
221 }
222 }
223 }
224
225 private void readPdtaChunk(RIFFReader riff) throws IOException {
226
227     List<SF2Instrument> presets = new ArrayList<SF2Instrument>();
228     List<Integer> presets_bagNdx = new ArrayList<Integer>();
229     List<SF2InstrumentRegion> presets_splits_gen
230         = new ArrayList<SF2InstrumentRegion>();
231     List<SF2InstrumentRegion> presets_splits_mod
232         = new ArrayList<SF2InstrumentRegion>();
233
234     List<SF2Layer> instruments = new ArrayList<SF2Layer>();
235     List<Integer> instruments_bagNdx = new ArrayList<Integer>();
236     List<SF2LayerRegion> instruments_splits_gen
237         = new ArrayList<SF2LayerRegion>();
238     List<SF2LayerRegion> instruments_splits_mod
239         = new ArrayList<SF2LayerRegion>();
240
241     while (riff.hasNextChunk()) {
242         RIFFReader chunk = riff.nextChunk();
243         String format = chunk.getFormat();
244         if (format.equals("phdr")) {
245             // Preset Header / Instrument
246             if (chunk.available() % 38 != 0)

```

```

247         throw new RIFFInvalidDataException();
248     int count = chunk.available() / 38;
249     for (int i = 0; i < count; i++) {
250         SF2Instrument preset = new SF2Instrument(this);
251         preset.name = chunk.readString(20);
252         preset.preset = chunk.readUnsignedShort();
253         preset.bank = chunk.readUnsignedShort();
254         presets_bagNdx.add(chunk.readUnsignedShort());
255         preset.library = chunk.readUnsignedInt();
256         preset.genre = chunk.readUnsignedInt();
257         preset.morphology = chunk.readUnsignedInt();
258         presets.add(preset);
259         if (i != count - 1)
260             this.instruments.add(preset);
261     }
262 } else if (format.equals("pbag")) {
263     // Preset Zones / Instruments splits
264     if (chunk.available() % 4 != 0)
265         throw new RIFFInvalidDataException();
266     int count = chunk.available() / 4;
267
268     // Skip first record
269     {
270         int gencount = chunk.readUnsignedShort();
271         int modcount = chunk.readUnsignedShort();
272         while (presets_splits_gen.size() < gencount)
273             presets_splits_gen.add(null);
274         while (presets_splits_mod.size() < modcount)
275             presets_splits_mod.add(null);
276         count--;
277     }
278
279     int offset = presets_bagNdx.get(0);
280     // Offset should be 0 (but just case)
281     for (int i = 0; i < offset; i++) {
282         if (count == 0)
283             throw new RIFFInvalidDataException();
284         int gencount = chunk.readUnsignedShort();
285         int modcount = chunk.readUnsignedShort();
286         while (presets_splits_gen.size() < gencount)
287             presets_splits_gen.add(null);
288         while (presets_splits_mod.size() < modcount)
289             presets_splits_mod.add(null);
290         count--;
291     }
292
293     for (int i = 0; i < presets_bagNdx.size() - 1; i++) {
294         int zone_count = presets_bagNdx.get(i + 1)
295             - presets_bagNdx.get(i);
296         SF2Instrument preset = presets.get(i);
297         for (int ii = 0; ii < zone_count; ii++) {
298             if (count == 0)
299                 throw new RIFFInvalidDataException();
300             int gencount = chunk.readUnsignedShort();
301             int modcount = chunk.readUnsignedShort();
302             SF2InstrumentRegion split = new SF2InstrumentRegion();
303             preset.regions.add(split);
304             while (presets_splits_gen.size() < gencount)
305                 presets_splits_gen.add(split);
306             while (presets_splits_mod.size() < modcount)
307                 presets_splits_mod.add(split);
308             count--;

```

```

309     }
310 }
311 } else if (format.equals("pmod")) {
312     // Preset Modulators / Split Modulators
313     for (int i = 0; i < presets_splits_mod.size(); i++) {
314         SF2Modulator modulator = new SF2Modulator();
315         modulator.sourceOperator = chunk.readUnsignedShort();
316         modulator.destinationOperator = chunk.readUnsignedShort();
317         modulator.amount = chunk.readShort();
318         modulator.amountSourceOperator = chunk.readUnsignedShort();
319         modulator.transportOperator = chunk.readUnsignedShort();
320         SF2InstrumentRegion split = presets_splits_mod.get(i);
321         if (split != null)
322             split.modulators.add(modulator);
323     }
324 } else if (format.equals("pgen")) {
325     // Preset Generators / Split Generators
326     for (int i = 0; i < presets_splits_gen.size(); i++) {
327         int operator = chunk.readUnsignedShort();
328         short amount = chunk.readShort();
329         SF2InstrumentRegion split = presets_splits_gen.get(i);
330         if (split != null)
331             split.generators.put(operator, amount);
332     }
333 } else if (format.equals("inst")) {
334     // Instrument Header / Layers
335     if (chunk.available() % 22 != 0)
336         throw new RIFFInvalidDataException();
337     int count = chunk.available() / 22;
338     for (int i = 0; i < count; i++) {
339         SF2Layer layer = new SF2Layer(this);
340         layer.name = chunk.readString(20);
341         instruments_bagNdx.add(chunk.readUnsignedShort());
342         instruments.add(layer);
343         if (i != count - 1)
344             this.layers.add(layer);
345     }
346 } else if (format.equals("ibag")) {
347     // Instrument Zones / Layer splits
348     if (chunk.available() % 4 != 0)
349         throw new RIFFInvalidDataException();
350     int count = chunk.available() / 4;
351
352     // Skip first record
353     {
354         int gencount = chunk.readUnsignedShort();
355         int modcount = chunk.readUnsignedShort();
356         while (instruments_splits_gen.size() < gencount)
357             instruments_splits_gen.add(null);
358         while (instruments_splits_mod.size() < modcount)
359             instruments_splits_mod.add(null);
360         count--;
361     }
362
363     int offset = instruments_bagNdx.get(0);
364     // Offset should be 0 (but just case)
365     for (int i = 0; i < offset; i++) {
366         if (count == 0)
367             throw new RIFFInvalidDataException();
368         int gencount = chunk.readUnsignedShort();
369         int modcount = chunk.readUnsignedShort();
370         while (instruments_splits_gen.size() < gencount)

```



```

371         instruments_splits_gen.add(null);
372     while (instruments_splits_mod.size() < modcount)
373         instruments_splits_mod.add(null);
374     count--;
375 }
376
377 for (int i = 0; i < instruments_bagNdx.size() - 1; i++) {
378     int zone_count = instruments_bagNdx.get(i + 1) - instruments_bagNdx.get(i);
379     SF2Layer layer = layers.get(i);
380     for (int ii = 0; ii < zone_count; ii++) {
381         if (count == 0)
382             throw new RIFFInvalidDataException();
383         int gencount = chunk.readUnsignedShort();
384         int modcount = chunk.readUnsignedShort();
385         SF2LayerRegion split = new SF2LayerRegion();
386         layer.regions.add(split);
387         while (instruments_splits_gen.size() < gencount)
388             instruments_splits_gen.add(split);
389         while (instruments_splits_mod.size() < modcount)
390             instruments_splits_mod.add(split);
391         count--;
392     }
393 }
394
395 } else if (format.equals("imod")) {
396     // Instrument Modulators / Split Modulators
397     for (int i = 0; i < instruments_splits_mod.size(); i++) {
398         SF2Modulator modulator = new SF2Modulator();
399         modulator.sourceOperator = chunk.readUnsignedShort();
400         modulator.destinationOperator = chunk.readUnsignedShort();
401         modulator.amount = chunk.readShort();
402         modulator.amountSourceOperator = chunk.readUnsignedShort();
403         modulator.transportOperator = chunk.readUnsignedShort();
404         SF2LayerRegion split = instruments_splits_gen.get(i);
405         if (split != null)
406             split.modulators.add(modulator);
407     }
408 } else if (format.equals("igen")) {
409     // Instrument Generators / Split Generators
410     for (int i = 0; i < instruments_splits_gen.size(); i++) {
411         int operator = chunk.readUnsignedShort();
412         short amount = chunk.readShort();
413         SF2LayerRegion split = instruments_splits_gen.get(i);
414         if (split != null)
415             split.generators.put(operator, amount);
416     }
417 } else if (format.equals("shdr")) {
418     // Sample Headers
419     if (chunk.available() % 46 != 0)
420         throw new RIFFInvalidDataException();
421     int count = chunk.available() / 46;
422     for (int i = 0; i < count; i++) {
423         SF2Sample sample = new SF2Sample(this);
424         sample.name = chunk.readString(20);
425         long start = chunk.readUnsignedInt();
426         long end = chunk.readUnsignedInt();
427         sample.data = sampleData.subbuffer(start * 2, end * 2, true);
428         if (sampleData24 != null)
429             sample.data24 = sampleData24.subbuffer(start, end, true);
430         /*
431         sample.data = new ModelByteBuffer(sampleData, (int)(start*2),
432             (int)((end - start)*2));

```

```

433         if (sampleData24 != null)
434             sample.data24 = new ModelByteBuffer(sampleData24,
435                 (int)start, (int)(end - start));
436
437         */
438         sample.startLoop = chunk.readUnsignedInt() - start;
439         sample.endLoop = chunk.readUnsignedInt() - start;
440         if (sample.startLoop < 0)
441             sample.startLoop = -1;
442         if (sample.endLoop < 0)
443             sample.endLoop = -1;
444         sample.sampleRate = chunk.readUnsignedInt();
445         sample.originalPitch = chunk.readUnsignedByte();
446         sample.pitchCorrection = chunk.readByte();
447         sample.sampleLink = chunk.readUnsignedShort();
448         sample.sampleType = chunk.readUnsignedShort();
449         if (i != count - 1)
450             this.samples.add(sample);
451     }
452 }
453
454 Iterator<SF2Layer> liter = this.layers.iterator();
455 while (liter.hasNext()) {
456     SF2Layer layer = liter.next();
457     Iterator<SF2LayerRegion> siter = layer.regions.iterator();
458     SF2Region globalsplit = null;
459     while (siter.hasNext()) {
460         SF2LayerRegion split = siter.next();
461         if (split.generators.get(SF2LayerRegion.GENERATOR_SAMPLEID) != null) {
462             int sampleid = split.generators.get(
463                 SF2LayerRegion.GENERATOR_SAMPLEID);
464             split.generators.remove(SF2LayerRegion.GENERATOR_SAMPLEID);
465             split.sample = samples.get(sampleid);
466         } else {
467             globalsplit = split;
468         }
469     }
470     if (globalsplit != null) {
471         layer.getRegions().remove(globalsplit);
472         SF2GlobalRegion gsplit = new SF2GlobalRegion();
473         gsplit.generators = globalsplit.generators;
474         gsplit.modulators = globalsplit.modulators;
475         layer.setGlobalZone(gsplit);
476     }
477 }
478
479
480 Iterator<SF2Instrument> iiter = this.instruments.iterator();
481 while (iiter.hasNext()) {
482     SF2Instrument instrument = iiter.next();
483     Iterator<SF2InstrumentRegion> siter = instrument.regions.iterator();
484     SF2Region globalsplit = null;
485     while (siter.hasNext()) {
486         SF2InstrumentRegion split = siter.next();
487         if (split.generators.get(SF2LayerRegion.GENERATOR_INSTRUMENT) != null) {
488             int instrumentid = split.generators.get(
489                 SF2InstrumentRegion.GENERATOR_INSTRUMENT);
490             split.generators.remove(SF2LayerRegion.GENERATOR_INSTRUMENT);
491             split.layer = layers.get(instrumentid);
492         } else {
493             globalsplit = split;
494         }

```

```

495     }
496
497     if (globalsplit != null) {
498         instrument.getRegions().remove(globalsplit);
499         SF2GlobalRegion gsplit = new SF2GlobalRegion();
500         gsplit.generators = globalsplit.generators;
501         gsplit.modulators = globalsplit.modulators;
502         instrument.setGlobalZone(gsplit);
503     }
504 }
505
506 }
507
508 public void save(String name) throws IOException {
509     writeSoundbank(new RIFFWriter(name, "sfbk"));
510 }
511
512 public void save(File file) throws IOException {
513     writeSoundbank(new RIFFWriter(file, "sfbk"));
514 }
515
516 public void save(OutputStream out) throws IOException {
517     writeSoundbank(new RIFFWriter(out, "sfbk"));
518 }
519
520 private void writeSoundbank(RIFFWriter writer) throws IOException {
521     writeInfo(writer.writeList("INFO"));
522     writeSdtaChunk(writer.writeList("sdta"));
523     writePdtaChunk(writer.writeList("pdta"));
524     writer.close();
525 }
526
527 private void writeInfoStringChunk(RIFFWriter writer, String name,
528     String value) throws IOException {
529     if (value == null)
530         return;
531     RIFFWriter chunk = writer.writeChunk(name);
532     chunk.writeString(value);
533     int len = value.getBytes("ascii").length;
534     chunk.write(0);
535     len++;
536     if (len % 2 != 0)
537         chunk.write(0);
538 }
539
540 private void writeInfo(RIFFWriter writer) throws IOException {
541     if (this.targetEngine == null)
542         this.targetEngine = "EMU8000";
543     if (this.name == null)
544         this.name = "";
545
546     RIFFWriter ifil_chunk = writer.writeChunk("ifil");
547     ifil_chunk.writeUnsignedShort(this.major);
548     ifil_chunk.writeUnsignedShort(this.minor);
549     writeInfoStringChunk(writer, "isng", this.targetEngine);
550     writeInfoStringChunk(writer, "INAM", this.name);
551     writeInfoStringChunk(writer, "irom", this.romName);
552     if (romVersionMajor != -1) {
553         RIFFWriter iver_chunk = writer.writeChunk("iver");
554         iver_chunk.writeUnsignedShort(this.romVersionMajor);
555         iver_chunk.writeUnsignedShort(this.romVersionMinor);
556     }

```

```

557 writeInfoStringChunk(writer, "ICRD", this.creationDate);
558 writeInfoStringChunk(writer, "IENG", this.engineers);
559 writeInfoStringChunk(writer, "IPRD", this.product);
560 writeInfoStringChunk(writer, "ICOP", this.copyright);
561 writeInfoStringChunk(writer, "ICMT", this.comments);
562 writeInfoStringChunk(writer, "ISFT", this.tools);
563
564 writer.close();
565 }
566
567 private void writeSdtaChunk(RIFFWriter writer) throws IOException {
568
569     byte[] pad = new byte[32];
570
571     RIFFWriter smpl_chunk = writer.writeChunk("smpl");
572     for (SF2Sample sample : samples) {
573         ModelByteBuffer data = sample.getDataBuffer();
574         data.writeTo(smpl_chunk);
575         /*
576         smpl_chunk.write(data.array(),
577             data.arrayOffset(),
578             data.capacity());
579         */
580         smpl_chunk.write(pad);
581         smpl_chunk.write(pad);
582     }
583     if (major < 2)
584         return;
585     if (major == 2 && minor < 4)
586         return;
587
588     for (SF2Sample sample : samples) {
589         ModelByteBuffer data24 = sample.getData24Buffer();
590         if (data24 == null)
591             return;
592     }
593
594     RIFFWriter sm24_chunk = writer.writeChunk("sm24");
595     for (SF2Sample sample : samples) {
596         ModelByteBuffer data = sample.getData24Buffer();
597         data.writeTo(sm24_chunk);
598         /*
599         sm24_chunk.write(data.array(),
600             data.arrayOffset(),
601             data.capacity());*/
602         smpl_chunk.write(pad);
603     }
604 }
605
606 private void writeModulators(RIFFWriter writer, List<SF2Modulator> modulators)
607     throws IOException {
608     for (SF2Modulator modulator : modulators) {
609         writer.writeUnsignedShort(modulator.sourceOperator);
610         writer.writeUnsignedShort(modulator.destinationOperator);
611         writer.writeShort(modulator.amount);
612         writer.writeUnsignedShort(modulator.amountSourceOperator);
613         writer.writeUnsignedShort(modulator.transportOperator);
614     }
615 }
616
617 private void writeGenerators(RIFFWriter writer, Map<Integer, Short> generators)

```

```

619         throws IOException {
620     Short keyrange = (Short) generators.get(SF2Region.GENERATOR_KEYRANGE);
621     Short velrange = (Short) generators.get(SF2Region.GENERATOR_VELRANGE);
622     if (keyrange != null) {
623         writer.writeUnsignedShort(SF2Region.GENERATOR_KEYRANGE);
624         writer.writeShort(keyrange);
625     }
626     if (velrange != null) {
627         writer.writeUnsignedShort(SF2Region.GENERATOR_VELRANGE);
628         writer.writeShort(velrange);
629     }
630     for (Map.Entry<Integer, Short> generator : generators.entrySet()) {
631         if (generator.getKey() == SF2Region.GENERATOR_KEYRANGE)
632             continue;
633         if (generator.getKey() == SF2Region.GENERATOR_VELRANGE)
634             continue;
635         writer.writeUnsignedShort(generator.getKey());
636         writer.writeShort(generator.getValue());
637     }
638 }
639
640 private void writePdtaChunk(RIFFWriter writer) throws IOException {
641
642     RIFFWriter phdr_chunk = writer.writeChunk("phdr");
643     int phdr_zone_count = 0;
644     for (SF2Instrument preset : this.instruments) {
645         phdr_chunk.writeString(preset.name, 20);
646         phdr_chunk.writeUnsignedShort(preset.preset);
647         phdr_chunk.writeUnsignedShort(preset.bank);
648         phdr_chunk.writeUnsignedShort(phdr_zone_count);
649         if (preset.getGlobalRegion() != null)
650             phdr_zone_count += 1;
651         phdr_zone_count += preset.getRegions().size();
652         phdr_chunk.writeUnsignedInt(preset.library);
653         phdr_chunk.writeUnsignedInt(preset.genre);
654         phdr_chunk.writeUnsignedInt(preset.morphology);
655     }
656     phdr_chunk.writeString("EOP", 20);
657     phdr_chunk.writeUnsignedShort(0);
658     phdr_chunk.writeUnsignedShort(0);
659     phdr_chunk.writeUnsignedShort(phdr_zone_count);
660     phdr_chunk.writeUnsignedInt(0);
661     phdr_chunk.writeUnsignedInt(0);
662     phdr_chunk.writeUnsignedInt(0);
663
664
665     RIFFWriter pbag_chunk = writer.writeChunk("pbag");
666     int pbag_gencount = 0;
667     int pbag_modcount = 0;
668     for (SF2Instrument preset : this.instruments) {
669         if (preset.getGlobalRegion() != null) {
670             pbag_chunk.writeUnsignedShort(pbag_gencount);
671             pbag_chunk.writeUnsignedShort(pbag_modcount);
672             pbag_gencount += preset.getGlobalRegion().getGenerators().size();
673             pbag_modcount += preset.getGlobalRegion().getModulators().size();
674         }
675         for (SF2InstrumentRegion region : preset.getRegions()) {
676             pbag_chunk.writeUnsignedShort(pbag_gencount);
677             pbag_chunk.writeUnsignedShort(pbag_modcount);
678             if (layers.indexOf(region.layer) != -1) {
679                 // One generator is used to reference to instrument record
680                 pbag_gencount += 1;

```

```

681     }
682     pbag_gencount += region.getGenerators().size();
683     pbag_modcount += region.getModulators().size();
684 }
685 }
686 }
687 pbag_chunk.writeUnsignedShort(pbag_gencount);
688 pbag_chunk.writeUnsignedShort(pbag_modcount);
689
690 RIFFWriter pmod_chunk = writer.writeChunk("pmod");
691 for (SF2Instrument preset : this.instruments) {
692     if (preset.getGlobalRegion() != null) {
693         writeModulators(pmod_chunk,
694             preset.getGlobalRegion().getModulators());
695     }
696     for (SF2InstrumentRegion region : preset.getRegions())
697         writeModulators(pmod_chunk, region.getModulators());
698 }
699 pmod_chunk.write(new byte[10]);
700
701 RIFFWriter pgen_chunk = writer.writeChunk("pgen");
702 for (SF2Instrument preset : this.instruments) {
703     if (preset.getGlobalRegion() != null) {
704         writeGenerators(pgen_chunk,
705             preset.getGlobalRegion().getGenerators());
706     }
707     for (SF2InstrumentRegion region : preset.getRegions()) {
708         writeGenerators(pgen_chunk, region.getGenerators());
709         int ix = (int) layers.indexOf(region.layer);
710         if (ix != -1) {
711             pgen_chunk.writeUnsignedShort(SF2Region.GENERATOR_INSTRUMENT);
712             pgen_chunk.writeShort((short) ix);
713         }
714     }
715 }
716 pgen_chunk.write(new byte[4]);
717
718 RIFFWriter inst_chunk = writer.writeChunk("inst");
719 int inst_zone_count = 0;
720 for (SF2Layer instrument : this.layers) {
721     inst_chunk.writeString(instrument.name, 20);
722     inst_chunk.writeUnsignedShort(inst_zone_count);
723     if (instrument.getGlobalRegion() != null)
724         inst_zone_count += 1;
725     inst_zone_count += instrument.getRegions().size();
726 }
727 inst_chunk.writeString("EOI", 20);
728 inst_chunk.writeUnsignedShort(inst_zone_count);
729
730
731 RIFFWriter ibag_chunk = writer.writeChunk("ibag");
732 int ibag_gencount = 0;
733 int ibag_modcount = 0;
734 for (SF2Layer instrument : this.layers) {
735     if (instrument.getGlobalRegion() != null) {
736         ibag_chunk.writeUnsignedShort(ibag_gencount);
737         ibag_chunk.writeUnsignedShort(ibag_modcount);
738         ibag_gencount
739             += instrument.getGlobalRegion().getGenerators().size();
740         ibag_modcount
741             += instrument.getGlobalRegion().getModulators().size();
742     }

```

```

743     for (SF2LayerRegion region : instrument.getRegions()) {
744         ibag_chunk.writeUnsignedShort(ibag_gencount);
745         ibag_chunk.writeUnsignedShort(ibag_modcount);
746         if (samples.indexOf(region.sample) != -1) {
747             // One generator is used to reference to instrument record
748             ibag_gencount += 1;
749         }
750         ibag_gencount += region.getGenerators().size();
751         ibag_modcount += region.getModulators().size();
752     }
753 }
754
755 ibag_chunk.writeUnsignedShort(ibag_gencount);
756 ibag_chunk.writeUnsignedShort(ibag_modcount);
757
758
759 RIFFWriter imod_chunk = writer.writeChunk("imod");
760 for (SF2Layer instrument : this.layers) {
761     if (instrument.getGlobalRegion() != null) {
762         writeModulators(imod_chunk,
763             instrument.getGlobalRegion().getModulators());
764     }
765     for (SF2LayerRegion region : instrument.getRegions())
766         writeModulators(imod_chunk, region.getModulators());
767 }
768 imod_chunk.write(new byte[10]);
769
770 RIFFWriter igen_chunk = writer.writeChunk("igen");
771 for (SF2Layer instrument : this.layers) {
772     if (instrument.getGlobalRegion() != null) {
773         writeGenerators(igen_chunk,
774             instrument.getGlobalRegion().getGenerators());
775     }
776     for (SF2LayerRegion region : instrument.getRegions()) {
777         writeGenerators(igen_chunk, region.getGenerators());
778         int ix = samples.indexOf(region.sample);
779         if (ix != -1) {
780             igen_chunk.writeUnsignedShort(SF2Region.GENERATOR_SAMPLEID);
781             igen_chunk.writeShort((short) ix);
782         }
783     }
784 }
785 igen_chunk.write(new byte[4]);
786
787
788 RIFFWriter shdr_chunk = writer.writeChunk("shdr");
789 long sample_pos = 0;
790 for (SF2Sample sample : samples) {
791     shdr_chunk.writeString(sample.name, 20);
792     long start = sample_pos;
793     sample_pos += sample.data.capacity() / 2;
794     long end = sample_pos;
795     long startLoop = sample.startLoop + start;
796     long endLoop = sample.endLoop + start;
797     if (startLoop < start)
798         startLoop = start;
799     if (endLoop > end)
800         endLoop = end;
801     shdr_chunk.writeUnsignedInt(start);
802     shdr_chunk.writeUnsignedInt(end);
803     shdr_chunk.writeUnsignedInt(startLoop);
804     shdr_chunk.writeUnsignedInt(endLoop);

```

```

805         shdr_chunk.writeUnsignedInt(sample.sampleRate);
806         shdr_chunk.writeUnsignedByte(sample.originalPitch);
807         shdr_chunk.writeByte(sample.pitchCorrection);
808         shdr_chunk.writeUnsignedShort(sample.sampleLink);
809         shdr_chunk.writeUnsignedShort(sample.sampleType);
810         sample_pos += 32;
811     }
812     shdr_chunk.writeString("EOS", 20);
813     shdr_chunk.write(new byte[26]);
814
815 }
816
817 public String getName() {
818     return name;
819 }
820
821 public String getVersion() {
822     return major + "." + minor;
823 }
824
825 public String getVendor() {
826     return engineers;
827 }
828
829 public String getDescription() {
830     return comments;
831 }
832
833 public void setName(String s) {
834     name = s;
835 }
836
837 public void setVendor(String s) {
838     engineers = s;
839 }
840
841 public void setDescription(String s) {
842     comments = s;
843 }
844
845 public SoundbankResource[] getResources() {
846     SoundbankResource[] resources
847         = new SoundbankResource[layers.size() + samples.size()];
848     int j = 0;
849     for (int i = 0; i < layers.size(); i++)
850         resources[j++] = layers.get(i);
851     for (int i = 0; i < samples.size(); i++)
852         resources[j++] = samples.get(i);
853     return resources;
854 }
855
856 public SF2Instrument[] getInstruments() {
857     SF2Instrument[] inslist_array
858         = instruments.toArray(new SF2Instrument[instruments.size()]);
859     Arrays.sort(inslist_array, new ModelInstrumentComparator());
860     return inslist_array;
861 }
862
863 public SF2Layer[] getLayers() {
864     return layers.toArray(new SF2Layer[layers.size()]);
865 }
866

```



```

867 public SF2Sample[] getSamples() {
868     return samples.toArray(new SF2Sample[samples.size()]);
869 }
870
871 public Instrument getInstrument(Patch patch) {
872     int program = patch.getProgram();
873     int bank = patch.getBank();
874     boolean percussion = false;
875     if (patch instanceof ModelPatch)
876         percussion = ((ModelPatch)patch).isPercussion();
877     for (Instrument instrument : instruments) {
878         Patch patch2 = instrument.getPatch();
879         int program2 = patch2.getProgram();
880         int bank2 = patch2.getBank();
881         if (program == program2 && bank == bank2) {
882             boolean percussion2 = false;
883             if (patch2 instanceof ModelPatch)
884                 percussion2 = ((ModelPatch) patch2).isPercussion();
885             if (percussion == percussion2)
886                 return instrument;
887         }
888     }
889     return null;
890 }
891
892 public String getCreationDate() {
893     return creationDate;
894 }
895
896 public void setCreationDate(String creationDate) {
897     this.creationDate = creationDate;
898 }
899
900 public String getProduct() {
901     return product;
902 }
903
904 public void setProduct(String product) {
905     this.product = product;
906 }
907
908 public String getRomName() {
909     return romName;
910 }
911
912 public void setRomName(String romName) {
913     this.romName = romName;
914 }
915
916 public int getRomVersionMajor() {
917     return romVersionMajor;
918 }
919
920 public void setRomVersionMajor(int romVersionMajor) {
921     this.romVersionMajor = romVersionMajor;
922 }
923
924 public int getRomVersionMinor() {
925     return romVersionMinor;
926 }
927
928 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

```
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.io.File;
28 import java.io.IOException;
29 import java.io.InputStream;
30 import java.net.URL;
31 import javax.sound.midi.InvalidMidiDataException;
32 import javax.sound.midi.Soundbank;
33 import javax.sound.midi.spi.SoundbankReader;
34
35 /**
36 * This class is used to connect the SF2SoundBank class
37 * to the SoundbankReader SPI interface.
38 *
39 * @author Karl Helgason
40 */
41 public class SF2SoundbankReader extends SoundbankReader {
42
43     public Soundbank getSoundbank(URL url)
44         throws InvalidMidiDataException, IOException {
45         try {
46             return new SF2Soundbank(url);
47         } catch (RIFFInvalidFormatException e) {
48             return null;
49         } catch (IOException ioe) {
50             return null;
51         }
52     }
53
54     public Soundbank getSoundbank(InputStream stream)
55         throws InvalidMidiDataException, IOException {
56         try {
57             stream.mark(512);
58             return new SF2Soundbank(stream);
59         } catch (RIFFInvalidFormatException e) {
60             stream.reset();
61         }
62     }
63 }
```

```
61         return null;
62     }
63 }
64
65 public Soundbank getSoundbank(File file)
66     throws InvalidMidiDataException, IOException {
67     try {
68         return new SF2Soundbank(file);
69     } catch (RIFFInvalidFormatException e) {
70         return null;
71     }
72 }
73 }
```

75 com/sun/media/sound/SimpleInstrument.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.ArrayList;
28 import java.util.List;
29 import javax.sound.midi.Patch;
30
31 /**
32  * A simple instrument that is made of other ModelInstrument, ModelPerformer
33  * objects.
34  *
35  * @author Karl Helgason
36  */
37 public class SimpleInstrument extends ModelInstrument {
38
39     private static class SimpleInstrumentPart {
40         ModelPerformer[] performers;
41         int keyFrom;
42         int keyTo;
43         int velFrom;
44         int velTo;
45         int exclusiveClass;
46     }
47     protected int preset = 0;
48     protected int bank = 0;
49     protected boolean percussion = false;
50     protected String name = "";
51     protected List<SimpleInstrumentPart> parts
52         = new ArrayList<SimpleInstrumentPart>();
53
54     public SimpleInstrument() {
55         super(null, null, null, null);
56     }
57
58     public void clear() {
59         parts.clear();
60     }
```

```

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 public ModelPerformer[] getPerformers() {
124
125     int percount = 0;
126     for (SimpleInstrumentPart part : parts)
127         if (part.performers != null)
128             percount += part.performers.length;
129
130     ModelPerformer[] performers = new ModelPerformer[percount];
131     int px = 0;
132     for (SimpleInstrumentPart part : parts) {
133         if (part.performers != null) {
134             for (ModelPerformer mperfm : part.performers) {
135                 ModelPerformer performer = new ModelPerformer();
136                 performer.setName(getName());
137                 performers[px++] = performer;
138
139                 performer.setDefaultConnectionsEnabled(
140                     mperfm.isDefaultConnectionsEnabled());
141                 performer.setKeyFrom(mperfm.getKeyFrom());
142                 performer.setKeyTo(mperfm.getKeyTo());
143                 performer.setVelFrom(mperfm.getVelFrom());
144                 performer.setVelTo(mperfm.getVelTo());
145                 performer.setExclusiveClass(mperfm.getExclusiveClass());
146                 performer.setSelfNonExclusive(mperfm.isSelfNonExclusive());
147                 performer.setReleaseTriggered(mperfm.isReleaseTriggered());
148                 if (part.exclusiveClass != -1)
149                     performer.setExclusiveClass(part.exclusiveClass);
150                 if (part.keyFrom > performer.getKeyFrom())
151                     performer.setKeyFrom(part.keyFrom);
152                 if (part.keyTo < performer.getKeyTo())
153                     performer.setKeyTo(part.keyTo);
154                 if (part.velFrom > performer.getVelFrom())
155                     performer.setVelFrom(part.velFrom);
156                 if (part.velTo < performer.getVelTo())
157                     performer.setVelTo(part.velTo);
158                 performer.getOscillators().addAll(mperfm.getOscillators());
159                 performer.getConnectionBlocks().addAll(
160                     mperfm.getConnectionBlocks());
161             }
162         }
163     }
164
165     return performers;
166 }
167
168 public Object getData() {
169     return null;
170 }
171
172 public String getName() {
173     return this.name;
174 }
175
176 public void setName(String name) {
177     this.name = name;
178 }
179
180 public ModelPatch getPatch() {
181     return new ModelPatch(bank, preset, percussion);
182 }
183
184

```

```
185     public void setPatch(Patch patch) {
186         if (patch instanceof ModelPatch && ((ModelPatch)patch).isPercussion()) {
187             percussion = true;
188             bank = patch.getBank();
189             preset = patch.getProgram();
190         } else {
191             percussion = false;
192             bank = patch.getBank();
193             preset = patch.getProgram();
194         }
195     }
196 }
```


76 com/sun/media/sound/SimpleSoundbank.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.ArrayList;
28 import java.util.Arrays;
29 import java.util.List;
30
31 import javax.sound.midi.Instrument;
32 import javax.sound.midi.Patch;
33 import javax.sound.midi.Soundbank;
34 import javax.sound.midi.SoundbankResource;
35
36 /**
37  * A simple soundbank that contains instruments and soundbankresources.
38  *
39  * @author Karl Helgason
40  */
41 public class SimpleSoundbank implements Soundbank {
42
43     String name = "";
44     String version = "";
45     String vendor = "";
46     String description = "";
47     List<SoundbankResource> resources = new ArrayList<SoundbankResource>();
48     List<Instrument> instruments = new ArrayList<Instrument>();
49
50     public String getName() {
51         return name;
52     }
53
54     public String getVersion() {
55         return version;
56     }
57
58     public String getVendor() {
59         return vendor;
60     }
```

```

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)

```

```

123         instruments.remove((Instrument) resource);
124     else
125         resources.remove(resource);
126 }
127
128 public void addInstrument(Instrument resource) {
129     instruments.add(resource);
130 }
131
132 public void removeInstrument(Instrument resource) {
133     instruments.remove(resource);
134 }
135
136 public void addAllInstruments(Soundbank soundbank) {
137     for (Instrument ins : soundbank.getInstruments())
138         addInstrument(ins);
139 }
140
141 public void removeAllInstruments(Soundbank soundbank) {
142     for (Instrument ins : soundbank.getInstruments())
143         removeInstrument(ins);
144 }
145 }

```

77 com/sun/media/sound/SoftAbstractResampler.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.io.IOException;
28 import java.util.Arrays;
29
30 import javax.sound.midi.MidiChannel;
31 import javax.sound.midi.VoiceStatus;
32
33 /**
34  * Abstract resampler class.
35  *
36  * @author Karl Helgason
37  */
38 public abstract class SoftAbstractResampler implements SoftResampler {
39
40     private class ModelAbstractResamplerStream implements SoftResamplerStreamer {
41
42         AudioFloatInputStream stream;
43         boolean stream_eof = false;
44         int loopmode;
45         boolean looppdirection = true; // true = forward
46         float loopstart;
47         float looplen;
48         float target_pitch;
49         float[] current_pitch = new float[1];
50         boolean started;
51         boolean eof;
52         int sector_pos = 0;
53         int sector_size = 400;
54         int sector_loopstart = -1;
55         boolean markset = false;
56         int marklimit = 0;
57         int streampos = 0;
58         int nrofchannels = 2;
59         boolean noteOff_flag = false;
60         float[][] ibuffer;
```

```

61     boolean ibuffer_order = true;
62     float[] sbuffer;
63     int pad;
64     int pad2;
65     float[] ix = new float[1];
66     int[] ox = new int[1];
67     float samplerateconv = 1;
68     float pitchcorrection = 0;
69
70     public ModelAbstractResamplerStream() {
71         pad = getPadding();
72         pad2 = getPadding() * 2;
73         ibuffer = new float[2][sector_size + pad2];
74         ibuffer_order = true;
75     }
76
77     public void noteOn(MidiChannel channel, VoiceStatus voice,
78         int noteNumber, int velocity) {
79     }
80
81     public void noteOff(int velocity) {
82         noteOff_flag = true;
83     }
84
85     public void open(ModelWavetable osc, float outputsamplerate)
86         throws IOException {
87
88         eof = false;
89         nrofchannels = osc.getChannels();
90         if (ibuffer.length < nrofchannels) {
91             ibuffer = new float[nrofchannels][sector_size + pad2];
92         }
93
94         stream = osc.openStream();
95         streampos = 0;
96         stream_eof = false;
97         pitchcorrection = osc.getPitchcorrection();
98         samplerateconv
99             = stream.getFormat().getSampleRate() / outputsamplerate;
100         looplen = osc.getLoopLength();
101         loopstart = osc.getLoopStart();
102         sector_loopstart = (int) (loopstart / sector_size);
103         sector_loopstart = sector_loopstart - 1;
104
105         sector_pos = 0;
106
107         if (sector_loopstart < 0)
108             sector_loopstart = 0;
109         started = false;
110         loopmode = osc.getLoopType();
111
112         if (loopmode != 0) {
113             markset = false;
114             marklimit = nrofchannels * (int) (looplen + pad2 + 1);
115         } else
116             markset = true;
117         // loopmode = 0;
118
119         target_pitch = samplerateconv;
120         current_pitch[0] = samplerateconv;
121
122         ibuffer_order = true;

```

```

123     loopdirection = true;
124     noteOff_flag = false;
125
126     for (int i = 0; i < nrofchannels; i++)
127         Arrays.fill(ibuffer[i], sector_size, sector_size + pad2, 0);
128     ix[0] = pad;
129     eof = false;
130
131     ix[0] = sector_size + pad;
132     sector_pos = -1;
133     streampos = -sector_size;
134
135     nextBuffer();
136 }
137
138 public void setPitch(float pitch) {
139     /*
140     this.pitch = (float) Math.pow(2f,
141     (pitchcorrection + pitch) / 1200.0f)
142     * samplerateconv;
143     */
144     this.target_pitch = (float) Math.exp(
145         (pitchcorrection + pitch) * (Math.log(2.0) / 1200.0))
146         * samplerateconv;
147
148     if (!started)
149         current_pitch[0] = this.target_pitch;
150 }
151
152 public void nextBuffer() throws IOException {
153     if (ix[0] < pad) {
154         if (markset) {
155             // reset to target sector
156             stream.reset();
157             ix[0] += streampos - (sector_loopstart * sector_size);
158             sector_pos = sector_loopstart;
159             streampos = sector_pos * sector_size;
160
161             // and go one sector backward
162             ix[0] += sector_size;
163             sector_pos -= 1;
164             streampos -= sector_size;
165             stream_eof = false;
166         }
167     }
168
169     if (ix[0] >= sector_size + pad) {
170         if (stream_eof) {
171             eof = true;
172             return;
173         }
174     }
175
176     if (ix[0] >= sector_size * 4 + pad) {
177         int skips = (int)((ix[0] - sector_size * 4 + pad) / sector_size);
178         ix[0] -= sector_size * skips;
179         sector_pos += skips;
180         streampos += sector_size * skips;
181         stream.skip(sector_size * skips);
182     }
183
184     while (ix[0] >= sector_size + pad) {

```

```

185     if (!markset) {
186         if (sector_pos + 1 == sector_loopstart) {
187             stream.mark(marklimit);
188             markset = true;
189         }
190     }
191     ix[0] -= sector_size;
192     sector_pos++;
193     streampos += sector_size;
194
195     for (int c = 0; c < nrofchannels; c++) {
196         float[] cbuffer = ibuffer[c];
197         for (int i = 0; i < pad2; i++)
198             cbuffer[i] = cbuffer[i + sector_size];
199     }
200
201     int ret;
202     if (nrofchannels == 1)
203         ret = stream.read(ibuffer[0], pad2, sector_size);
204     else {
205         int slen = sector_size * nrofchannels;
206         if (sbuffer == null || sbuffer.length < slen)
207             sbuffer = new float[slen];
208         int sret = stream.read(sbuffer, 0, slen);
209         if (sret == -1)
210             ret = -1;
211         else {
212             ret = sret / nrofchannels;
213             for (int i = 0; i < nrofchannels; i++) {
214                 float[] buff = ibuffer[i];
215                 int ix = i;
216                 int ix_step = nrofchannels;
217                 int ox = pad2;
218                 for (int j = 0; j < ret; j++, ix += ix_step, ox++)
219                     buff[ox] = sbuffer[ix];
220             }
221         }
222     }
223
224     if (ret == -1) {
225         ret = 0;
226         stream_eof = true;
227         for (int i = 0; i < nrofchannels; i++)
228             Arrays.fill(ibuffer[i], pad2, pad2 + sector_size, 0f);
229         return;
230     }
231     if (ret != sector_size) {
232         for (int i = 0; i < nrofchannels; i++)
233             Arrays.fill(ibuffer[i], pad2 + ret, pad2 + sector_size, 0f);
234     }
235
236     ibuffer_order = true;
237
238 }
239
240
241 }
242
243 public void reverseBuffers() {
244     ibuffer_order = !ibuffer_order;
245     for (int c = 0; c < nrofchannels; c++) {
246         float[] cbuff = ibuffer[c];

```

```

247     int len = cbuff.length - 1;
248     int len2 = cbuff.length / 2;
249     for (int i = 0; i < len2; i++) {
250         float x = cbuff[i];
251         cbuff[i] = cbuff[len - i];
252         cbuff[len - i] = x;
253     }
254 }
255 }
256
257 public int read(float[][] buffer, int offset, int len)
258     throws IOException {
259
260     if (eof)
261         return -1;
262
263     if (noteOff_flag)
264         if ((loopmode & 2) != 0)
265             if (loopdirection)
266                 loopmode = 0;
267
268     float pitchstep = (target_pitch - current_pitch[0]) / len;
269     float[] current_pitch = this.current_pitch;
270     started = true;
271
272     int[] ox = this.ox;
273     ox[0] = offset;
274     int ox_end = len + offset;
275
276     float ixend = sector_size + pad;
277     if (!loopdirection)
278         ixend = pad;
279     while (ox[0] != ox_end) {
280         nextBuffer();
281         if (!loopdirection) {
282             // If we are in backward playing part of pingpong
283             // or reverse loop
284
285             if (streampos < (loopstart + pad)) {
286                 ixend = loopstart - streampos + pad2;
287                 if (ix[0] <= ixend) {
288                     if ((loopmode & 4) != 0) {
289                         // Ping pong loop, change loopdirection
290                         loopdirection = true;
291                         ixend = sector_size + pad;
292                         continue;
293                     }
294
295                     ix[0] += loopplen;
296                     ixend = pad;
297                     continue;
298                 }
299             }
300         }
301
302         if (ibuffer_order != loopdirection)
303             reverseBuffers();
304
305         ix[0] = (sector_size + pad2) - ix[0];
306         ixend = (sector_size + pad2) - ixend;
307         ixend++;
308     }

```



```

309     float bak_ix = ix[0];
310     int bak_ox = ox[0];
311     float bak_pitch = current_pitch[0];
312     for (int i = 0; i < nrofchannels; i++) {
313         if (buffer[i] != null) {
314             ix[0] = bak_ix;
315             ox[0] = bak_ox;
316             current_pitch[0] = bak_pitch;
317             interpolate(ibuffer[i], ix, ixend, current_pitch,
318                       pitchstep, buffer[i], ox, ox_end);
319         }
320     }
321
322     ix[0] = (sector_size + pad2) - ix[0];
323     ixend--;
324     ixend = (sector_size + pad2) - ixend;
325
326     if (eof) {
327         current_pitch[0] = this.target_pitch;
328         return ox[0] - offset;
329     }
330
331     continue;
332 }
333 if (loopmode != 0) {
334     if (streampos + sector_size > (looplen + loopstart + pad)) {
335         ixend = loopstart + looplen - streampos + pad2;
336         if (ix[0] >= ixend) {
337             if ((loopmode & 4) != 0 || (loopmode & 8) != 0) {
338                 // Ping pong or reverse loop, change loopdirection
339                 loopdirection = false;
340                 ixend = pad;
341                 continue;
342             }
343             ixend = sector_size + pad;
344             ix[0] -= looplen;
345             continue;
346         }
347     }
348 }
349
350 if (ibuffer_order != loopdirection)
351     reverseBuffers();
352
353 float bak_ix = ix[0];
354 int bak_ox = ox[0];
355 float bak_pitch = current_pitch[0];
356 for (int i = 0; i < nrofchannels; i++) {
357     if (buffer[i] != null) {
358         ix[0] = bak_ix;
359         ox[0] = bak_ox;
360         current_pitch[0] = bak_pitch;
361         interpolate(ibuffer[i], ix, ixend, current_pitch,
362                   pitchstep, buffer[i], ox, ox_end);
363     }
364 }
365
366 if (eof) {
367     current_pitch[0] = this.target_pitch;
368     return ox[0] - offset;
369 }
370 }

```

```
371         current_pitch[0] = this.target_pitch;
372         return len;
373     }
374
375     public void close() throws IOException {
376         stream.close();
377     }
378 }
379
380 public abstract int getPadding();
381
382 public abstract void interpolate(float[] in, float[] in_offset,
383     float in_end, float[] pitch, float pitchstep, float[] out,
384     int[] out_offset, int out_end);
385
386 public SoftResamplerStreamer openStreamer() {
387     return new ModelAbstractResamplerStream();
388 }
389
390 }
```

78 com/sun/media/sound/SoftAudioBuffer.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.Arrays;
28
29 import javax.sound.sampled.AudioFormat;
30
31 /**
32  * This class is used to store audio buffer.
33  *
34  * @author Karl Helgason
35  */
36 public class SoftAudioBuffer {
37
38     private int size;
39     private float[] buffer;
40     private boolean empty = true;
41     private AudioFormat format;
42     private AudioFloatConverter converter;
43     private byte[] converter_buffer;
44
45     public SoftAudioBuffer(int size, AudioFormat format) {
46         this.size = size;
47         this.format = format;
48         converter = AudioFloatConverter.getConverter(format);
49     }
50
51     public void swap(SoftAudioBuffer swap)
52     {
53         int bak_size = size;
54         float[] bak_buffer = buffer;
55         boolean bak_empty = empty;
56         AudioFormat bak_format = format;
57         AudioFloatConverter bak_converter = converter;
58         byte[] bak_converter_buffer = converter_buffer;
59
60         size = swap.size;
```

```

61     buffer = swap.buffer;
62     empty = swap.empty;
63     format = swap.format;
64     converter = swap.converter;
65     converter_buffer = swap.converter_buffer;
66
67     swap.size = bak_size;
68     swap.buffer = bak_buffer;
69     swap.empty = bak_empty;
70     swap.format = bak_format;
71     swap.converter = bak_converter;
72     swap.converter_buffer = bak_converter_buffer;
73 }
74
75 public AudioFormat getFormat() {
76     return format;
77 }
78
79 public int getSize() {
80     return size;
81 }
82
83 public void clear() {
84     if (!empty) {
85         Arrays.fill(buffer, 0);
86         empty = true;
87     }
88 }
89
90 public boolean isSilent() {
91     return empty;
92 }
93
94 public float[] array() {
95     empty = false;
96     if (buffer == null)
97         buffer = new float[size];
98     return buffer;
99 }
100
101 public void get(byte[] buffer, int channel) {
102
103     int framesize_pc = (format.getFrameSize() / format.getChannels());
104     int c_len = size * framesize_pc;
105     if (converter_buffer == null || converter_buffer.length < c_len)
106         converter_buffer = new byte[c_len];
107
108     if (format.getChannels() == 1) {
109         converter.toByteArray(array(), size, buffer);
110     } else {
111         converter.toByteArray(array(), size, converter_buffer);
112         if (channel >= format.getChannels())
113             return;
114         int z_stepover = format.getChannels() * framesize_pc;
115         int k_stepover = framesize_pc;
116         for (int j = 0; j < framesize_pc; j++) {
117             int k = j;
118             int z = channel * framesize_pc + j;
119             for (int i = 0; i < size; i++) {
120                 buffer[z] = converter_buffer[k];
121                 z += z_stepover;
122                 k += k_stepover;

```

123 }
124 }
125 }
126 }
127 }
128 }

79 com/sun/media/sound/SoftAudioProcessor.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 /**
28  * Audio processor interface.
29  *
30  * @author Karl Helgason
31  */
32 public interface SoftAudioProcessor {
33
34     public void globalParameterControlChange(int[] slothpath, long param,
35         long value);
36
37     public void init(float samplerate, float controlrate);
38
39     public void setInput(int pin, SoftAudioBuffer input);
40
41     public void setOutput(int pin, SoftAudioBuffer output);
42
43     public void setMixMode(boolean mix);
44
45     public void processAudio();
46
47     public void processControlLogic();
48 }
```

80 com/sun/media/sound/SoftAudioPusher.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.io.IOException;
28
29 import javax.sound.sampled.AudioInputStream;
30 import javax.sound.sampled.SourceDataLine;
31
32 /**
33  * This is a processor object that writes into SourceDataLine
34  *
35  * @author Karl Helgason
36  */
37 public class SoftAudioPusher implements Runnable {
38
39     private volatile boolean active = false;
40     private SourceDataLine sourceDataLine = null;
41     private Thread audiothread;
42     private AudioInputStream ais;
43     private byte[] buffer;
44
45     public SoftAudioPusher(SourceDataLine sourceDataLine, AudioInputStream ais,
46         int workbuffersizer) {
47         this.ais = ais;
48         this.buffer = new byte[workbuffersizer];
49         this.sourceDataLine = sourceDataLine;
50     }
51
52     public synchronized void start() {
53         if (active)
54             return;
55         active = true;
56         audiothread = new Thread(this);
57         audiothread.setDaemon(true);
58         audiothread.setPriority(Thread.MAX_PRIORITY);
59         audiothread.start();
60     }
```

```

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

```
1  /*
2  * Copyright 2007-2010 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.ArrayList;
28 import java.util.Arrays;
29 import java.util.HashMap;
30 import java.util.Iterator;
31 import java.util.List;
32 import java.util.Map;
33
34 import javax.sound.midi.MidiChannel;
35 import javax.sound.midi.Patch;
36
37 /**
38  * Software Synthesizer MIDI channel class.
39  *
40  * @author Karl Helgason
41  */
42 public class SoftChannel implements MidiChannel, ModelDirectedPlayer {
43
44     private static boolean[] dontResetControls = new boolean[128];
45     static {
46         for (int i = 0; i < dontResetControls.length; i++)
47             dontResetControls[i] = false;
48
49         dontResetControls[0] = true; // Bank Select (MSB)
50         dontResetControls[32] = true; // Bank Select (LSB)
51         dontResetControls[7] = true; // Channel Volume (MSB)
52         dontResetControls[8] = true; // Balance (MSB)
53         dontResetControls[10] = true; // Pan (MSB)
54         dontResetControls[11] = true; // Expression (MSB)
55         dontResetControls[91] = true; // Effects 1 Depth (default: Reverb Send)
56         dontResetControls[92] = true; // Effects 2 Depth (default: Tremolo Depth)
57         dontResetControls[93] = true; // Effects 3 Depth (default: Chorus Send)
58         dontResetControls[94] = true; // Effects 4 Depth (default: Celeste [Detune] Depth)
59         dontResetControls[95] = true; // Effects 5 Depth (default: Phaser Depth)
60         dontResetControls[70] = true; // Sound Controller 1 (default: Sound Variation)
```

```

61     dontResetControls[71] = true; // Sound Controller 2 (default: Timbre / Harmonic Quality)
62     dontResetControls[72] = true; // Sound Controller 3 (default: Release Time)
63     dontResetControls[73] = true; // Sound Controller 4 (default: Attack Time)
64     dontResetControls[74] = true; // Sound Controller 5 (default: Brightness)
65     dontResetControls[75] = true; // Sound Controller 6 (GM2 default: Decay Time)
66     dontResetControls[76] = true; // Sound Controller 7 (GM2 default: Vibrato Rate)
67     dontResetControls[77] = true; // Sound Controller 8 (GM2 default: Vibrato Depth)
68     dontResetControls[78] = true; // Sound Controller 9 (GM2 default: Vibrato Delay)
69     dontResetControls[79] = true; // Sound Controller 10 (GM2 default: Undefined)
70     dontResetControls[120] = true; // All Sound Off
71     dontResetControls[121] = true; // Reset All Controllers
72     dontResetControls[122] = true; // Local Control On/Off
73     dontResetControls[123] = true; // All Notes Off
74     dontResetControls[124] = true; // Omni Mode Off
75     dontResetControls[125] = true; // Omni Mode On
76     dontResetControls[126] = true; // Poly Mode Off
77     dontResetControls[127] = true; // Poly Mode On
78
79     dontResetControls[6] = true; // Data Entry (MSB)
80     dontResetControls[38] = true; // Data Entry (LSB)
81     dontResetControls[96] = true; // Data Increment
82     dontResetControls[97] = true; // Data Decrement
83     dontResetControls[98] = true; // Non-Registered Parameter Number (LSB)
84     dontResetControls[99] = true; // Non-Registered Parameter Number (MSB)
85     dontResetControls[100] = true; // RPN = Null
86     dontResetControls[101] = true; // RPN = Null
87
88 }
89
90 private static final int RPN_NULL_VALUE = (127 << 7) + 127;
91 private int rpn_control = RPN_NULL_VALUE;
92 private int nrpn_control = RPN_NULL_VALUE;
93 protected double portamento_time = 1; // keychanges per control buffer time
94 protected int[] portamento_lastnote = new int[128];
95 protected int portamento_lastnote_ix = 0;
96 private boolean portamento = false;
97 private boolean mono = false;
98 private boolean mute = false;
99 private boolean solo = false;
100 private boolean solomute = false;
101 private Object control_mutex;
102 private int channel;
103 private SoftVoice[] voices;
104 private int bank;
105 private int program;
106 private SoftSynthesizer synthesizer;
107 private SoftMainMixer mainmixer;
108 private int[] polypressure = new int[128];
109 private int channelpressure = 0;
110 private int[] controller = new int[128];
111 private int pitchbend;
112 private double[] co_midi_pitch = new double[1];
113 private double[] co_midi_channel_pressure = new double[1];
114 protected SoftTuning tuning = new SoftTuning();
115 protected int tuning_bank = 0;
116 protected int tuning_program = 0;
117 protected SoftInstrument current_instrument = null;
118 protected ModelChannelMixer current_mixer = null;
119 protected ModelDirector current_director = null;
120
121 // Controller Destination Settings
122 protected int cds_control_number = -1;

```

```

123 protected ModelConnectionBlock[] cds_control_connections = null;
124 protected ModelConnectionBlock[] cds_channelpressure_connections = null;
125 protected ModelConnectionBlock[] cds_polypressure_connections = null;
126 protected boolean sustain = false;
127 protected boolean[][] keybasedcontroller_active = null;
128 protected double[][] keybasedcontroller_value = null;
129
130 private class MidiControlObject implements SoftControl {
131     double[] pitch = co_midi_pitch;
132     double[] channel_pressure = co_midi_channel_pressure;
133     double[] poly_pressure = new double[1];
134
135     public double[] get(int instance, String name) {
136         if (name == null)
137             return null;
138         if (name.equals("pitch"))
139             return pitch;
140         if (name.equals("channel_pressure"))
141             return channel_pressure;
142         if (name.equals("poly_pressure"))
143             return poly_pressure;
144         return null;
145     }
146 }
147
148 private SoftControl[] co_midi = new SoftControl[128];
149 {
150     for (int i = 0; i < co_midi.length; i++) {
151         co_midi[i] = new MidiControlObject();
152     }
153 }
154
155 private double[][] co_midi_cc_cc = new double[128][1];
156 private SoftControl co_midi_cc = new SoftControl() {
157     double[][] cc = co_midi_cc_cc;
158     public double[] get(int instance, String name) {
159         if (name == null)
160             return null;
161         return cc[Integer.parseInt(name)];
162     }
163 };
164 Map<Integer, int[]> co_midi_rpn_rpn_i = new HashMap<Integer, int[]>();
165 Map<Integer, double[]> co_midi_rpn_rpn = new HashMap<Integer, double[]>();
166 private SoftControl co_midi_rpn = new SoftControl() {
167     Map<Integer, double[]> rpn = co_midi_rpn_rpn;
168     public double[] get(int instance, String name) {
169         if (name == null)
170             return null;
171         int iname = Integer.parseInt(name);
172         double[] v = rpn.get(iname);
173         if (v == null) {
174             v = new double[1];
175             rpn.put(iname, v);
176         }
177         return v;
178     }
179 };
180 Map<Integer, int[]> co_midi_nrpn_nrpn_i = new HashMap<Integer, int[]>();
181 Map<Integer, double[]> co_midi_nrpn_nrpn = new HashMap<Integer, double[]>();
182 private SoftControl co_midi_nrpn = new SoftControl() {
183     Map<Integer, double[]> nrpn = co_midi_nrpn_nrpn;
184     public double[] get(int instance, String name) {

```

```

185         if (name == null)
186             return null;
187         int iname = Integer.parseInt(name);
188         double[] v = nrpn.get(iname);
189         if (v == null) {
190             v = new double[1];
191             nrpn.put(iname, v);
192         }
193         return v;
194     }
195 };
196
197 private static int restrict7Bit(int value)
198 {
199     if(value < 0) return 0;
200     if(value > 127) return 127;
201     return value;
202 }
203
204 private static int restrict14Bit(int value)
205 {
206     if(value < 0) return 0;
207     if(value > 16256) return 16256;
208     return value;
209 }
210
211 public SoftChannel(SoftSynthesizer synth, int channel) {
212     this.channel = channel;
213     this.voices = synth.getVoices();
214     this.synthesizer = synth;
215     this.mainmixer = synth.getMainMixer();
216     control_mutex = synth.control_mutex;
217     resetAllControllers(true);
218 }
219
220 private int findFreeVoice(int x) {
221     if(x == -1)
222     {
223         // x = -1 means that there where no available voice
224         // last time we called findFreeVoice
225         // and it hasn't changed because no audio has been
226         // rendered in the meantime.
227         // Therefore we have to return -1.
228         return -1;
229     }
230     for (int i = x; i < voices.length; i++)
231         if (!voices[i].active)
232             return i;
233
234     // No free voice was found, we must steal one
235
236     int vmode = synthesizer.getVoiceAllocationMode();
237     if (vmode == 1) {
238         // DLS Static Voice Allocation
239
240         // * priority ( 10, 1-9, 11-16)
241         // Search for channel to steal from
242         int steal_channel = channel;
243         for (int j = 0; j < voices.length; j++) {
244             if (voices[j].stealer_channel == null) {
245                 if (steal_channel == 9) {
246                     steal_channel = voices[j].channel;

```

```

247         } else {
248             if (voices[j].channel != 9) {
249                 if (voices[j].channel > steal_channel)
250                     steal_channel = voices[j].channel;
251             }
252         }
253     }
254 }
255
256 int voiceNo = -1;
257
258 SoftVoice v = null;
259 // Search for oldest voice in off state on steal_channel
260 for (int j = 0; j < voices.length; j++) {
261     if (voices[j].channel == steal_channel) {
262         if (voices[j].stealer_channel == null && !voices[j].on) {
263             if (v == null) {
264                 v = voices[j];
265                 voiceNo = j;
266             }
267             if (voices[j].voiceID < v.voiceID) {
268                 v = voices[j];
269                 voiceNo = j;
270             }
271         }
272     }
273 }
274 // Search for oldest voice in on state on steal_channel
275 if (voiceNo == -1) {
276     for (int j = 0; j < voices.length; j++) {
277         if (voices[j].channel == steal_channel) {
278             if (voices[j].stealer_channel == null) {
279                 if (v == null) {
280                     v = voices[j];
281                     voiceNo = j;
282                 }
283                 if (voices[j].voiceID < v.voiceID) {
284                     v = voices[j];
285                     voiceNo = j;
286                 }
287             }
288         }
289     }
290 }
291
292 return voiceNo;
293
294 } else {
295     // Default Voice Allocation
296     // * Find voice that is on
297     //     and Find voice which has lowest voiceID ( oldest voice)
298     // * Or find voice that is off
299     //     and Find voice which has lowest voiceID ( oldest voice)
300
301     int voiceNo = -1;
302
303     SoftVoice v = null;
304     // Search for oldest voice in off state
305     for (int j = 0; j < voices.length; j++) {
306         if (voices[j].stealer_channel == null && !voices[j].on) {
307             if (v == null) {
308                 v = voices[j];

```

```

309         voiceNo = j;
310     }
311     if (voices[j].voiceID < v.voiceID) {
312         v = voices[j];
313         voiceNo = j;
314     }
315 }
316 }
317 // Search for oldest voice in on state
318 if (voiceNo == -1) {
319
320     for (int j = 0; j < voices.length; j++) {
321         if (voices[j].stealer_channel == null) {
322             if (v == null) {
323                 v = voices[j];
324                 voiceNo = j;
325             }
326             if (voices[j].voiceID < v.voiceID) {
327                 v = voices[j];
328                 voiceNo = j;
329             }
330         }
331     }
332 }
333
334 return voiceNo;
335 }
336
337 }
338
339 protected void initVoice(SoftVoice voice, SoftPerformer p, int voiceID,
340     int noteNumber, int velocity, int delay, ModelConnectionBlock[] connectionBlocks,
341     ModelChannelMixer channelmixer, boolean releaseTriggered) {
342     if (voice.active) {
343         // Voice is active , we must steal the voice
344         voice.stealer_channel = this;
345         voice.stealer_performer = p;
346         voice.stealer_voiceID = voiceID;
347         voice.stealer_noteNumber = noteNumber;
348         voice.stealer_velocity = velocity;
349         voice.stealer_extendedConnectionBlocks = connectionBlocks;
350         voice.stealer_channelmixer = channelmixer;
351         voice.stealer_releaseTriggered = releaseTriggered;
352         for (int i = 0; i < voices.length; i++)
353             if (voices[i].active && voices[i].voiceID == voice.voiceID)
354                 voices[i].soundOff();
355         return;
356     }
357
358     voice.extendedConnectionBlocks = connectionBlocks;
359     voice.channelmixer = channelmixer;
360     voice.releaseTriggered = releaseTriggered;
361     voice.voiceID = voiceID;
362     voice.tuning = tuning;
363     voice.exclusiveClass = p.exclusiveClass;
364     voice.softchannel = this;
365     voice.channel = channel;
366     voice.bank = bank;
367     voice.program = program;
368     voice.instrument = current_instrument;
369     voice.performer = p;
370     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);

```



```

495         current_mixer = current_instrument.getSourceInstrument()
496             .getChannelMixer(this, synthesizer.getFormat());
497         if (current_mixer != null)
498             mainmixer.registerMixer(current_mixer);
499         current_director = current_instrument.getDirector(this, this);
500         applyInstrumentCustomization();
501     }
502     prevVoiceID = synthesizer.voiceIDCounter++;
503     firstVoice = true;
504     voiceNo = 0;
505
506     int tunedKey = (int)(Math.round(tuning.getTuning()[noteNumber]/100.0));
507     play_noteNumber = noteNumber;
508     play_velocity = velocity;
509     play_delay = delay;
510     play_releasetriggered = false;
511     lastVelocity[noteNumber] = velocity;
512     current_director.noteOn(tunedKey, velocity);
513
514     /*
515     SoftPerformer[] performers = current_instrument.getPerformers();
516     for (int i = 0; i < performers.length; i++) {
517         SoftPerformer p = performers[i];
518         if (p.keyFrom <= tunedKey && p.keyTo >= tunedKey) {
519             if (p.velFrom <= velocity && p.velTo >= velocity) {
520                 if (firstVoice) {
521                     firstVoice = false;
522                     if (p.exclusiveClass != 0) {
523                         int x = p.exclusiveClass;
524                         for (int j = 0; j < voices.length; j++) {
525                             if (voices[j].active
526                                 && voices[j].channel == channel
527                                 && voices[j].exclusiveClass == x) {
528                                 if (!(p.selfNonExclusive
529                                     && voices[j].note == noteNumber))
530                                     voices[j].shutdown();
531                             }
532                         }
533                     }
534                 }
535                 voiceNo = findFreeVoice(voiceNo);
536                 if (voiceNo == -1)
537                     return;
538                 initVoice(voices[voiceNo], p, prevVoiceID, noteNumber,
539                     velocity);
540             }
541         }
542     }
543     */
544 }
545
546
547 public void noteOff(int noteNumber, int velocity) {
548     noteNumber = restrict7Bit(noteNumber);
549     velocity = restrict7Bit(velocity);
550     noteOff_internal(noteNumber, velocity);
551
552     if (current_mixer != null)
553         current_mixer.noteOff(noteNumber, velocity);
554 }
555
556 private void noteOff_internal(int noteNumber, int velocity) {

```

```

557 synchronized (control_mutex) {
558
559     if (!mono) {
560         if (portamento) {
561             if (portamento_lastnote_ix != 127) {
562                 portamento_lastnote[portamento_lastnote_ix] = noteNumber;
563                 portamento_lastnote_ix++;
564             }
565         }
566     }
567
568     mainmixer.activity();
569     for (int i = 0; i < voices.length; i++) {
570         if (voices[i].on && voices[i].channel == channel
571             && voices[i].note == noteNumber
572             && voices[i].releaseTriggered == false) {
573             voices[i].noteOff(velocity);
574         }
575         // We must also check stolen voices
576         if (voices[i].stealer_channel == this && voices[i].stealer_noteNumber ==
577             noteNumber) {
578             SoftVoice v = voices[i];
579             v.stealer_releaseTriggered = false;
580             v.stealer_channel = null;
581             v.stealer_performer = null;
582             v.stealer_voiceID = -1;
583             v.stealer_noteNumber = 0;
584             v.stealer_velocity = 0;
585             v.stealer_extendedConnectionBlocks = null;
586             v.stealer_channelmixer = null;
587         }
588     }
589
590     // Try play back note-off triggered voices,
591
592     if (current_instrument == null) {
593         current_instrument
594             = synthesizer.findInstrument(program, bank, channel);
595         if (current_instrument == null)
596             return;
597         if (current_mixer != null)
598             mainmixer.stopMixer(current_mixer);
599         current_mixer = current_instrument.getSourceInstrument()
600             .getChannelMixer(this, synthesizer.getFormat());
601         if (current_mixer != null)
602             mainmixer.registerMixer(current_mixer);
603         current_director = current_instrument.getDirector(this, this);
604         applyInstrumentCustomization();
605     }
606     prevVoiceID = synthesizer.voiceIDCounter++;
607     firstVoice = true;
608     voiceNo = 0;
609
610     int tunedKey = (int)(Math.round(tuning.getTuning()[noteNumber]/100.0));
611     play_noteNumber = noteNumber;
612     play_velocity = lastVelocity[noteNumber];
613     play_releasetriggered = true;
614     play_delay = 0;
615     current_director.noteOff(tunedKey, velocity);
616
617 }

```

```

618 }
619 private int[] lastVelocity = new int[128];
620 private int prevVoiceID;
621 private boolean firstVoice = true;
622 private int voiceNo = 0;
623 private int play_noteNumber = 0;
624 private int play_velocity = 0;
625 private int play_delay = 0;
626 private boolean play_releasetriggered = false;
627
628 public void play(int performerIndex, ModelConnectionBlock[] connectionBlocks) {
629
630     int noteNumber = play_noteNumber;
631     int velocity = play_velocity;
632     int delay = play_delay;
633     boolean releasetriggered = play_releasetriggered;
634
635     SoftPerformer p = current_instrument.getPerformers()[performerIndex];
636
637     if (firstVoice) {
638         firstVoice = false;
639         if (p.exclusiveClass != 0) {
640             int x = p.exclusiveClass;
641             for (int j = 0; j < voices.length; j++) {
642                 if (voices[j].active && voices[j].channel == channel
643                     && voices[j].exclusiveClass == x) {
644                     if (!(p.selfNonExclusive && voices[j].note == noteNumber))
645                         voices[j].shutdown();
646                 }
647             }
648         }
649     }
650
651     voiceNo = findFreeVoice(voiceNo);
652
653     if (voiceNo == -1)
654         return;
655
656     initVoice(voices[voiceNo], p, prevVoiceID, noteNumber, velocity, delay,
657         connectionBlocks, current_mixer, releasetriggered);
658 }
659
660 public void noteOff(int noteNumber) {
661     if (noteNumber < 0 || noteNumber > 127) return;
662     noteOff_internal(noteNumber, 64);
663 }
664
665 public void setPolyPressure(int noteNumber, int pressure) {
666     noteNumber = restrict7Bit(noteNumber);
667     pressure = restrict7Bit(pressure);
668
669     if (current_mixer != null)
670         current_mixer.setPolyPressure(noteNumber, pressure);
671
672     synchronized (control_mutex) {
673         mainmixer.activity();
674         co_midi[noteNumber].get(0, "poly_pressure")[0] = pressure*(1.0/128.0);
675         polypressure[noteNumber] = pressure;
676         for (int i = 0; i < voices.length; i++) {
677             if (voices[i].active && voices[i].note == noteNumber)
678                 voices[i].setPolyPressure(pressure);
679         }

```

```

680     }
681 }
682
683 public int getPolyPressure(int noteNumber) {
684     synchronized (control_mutex) {
685         return polypressure[noteNumber];
686     }
687 }
688
689 public void setChannelPressure(int pressure) {
690     pressure = restrict7Bit(pressure);
691     if (current_mixer != null)
692         current_mixer.setChannelPressure(pressure);
693     synchronized (control_mutex) {
694         mainmixer.activity();
695         co_midi_channel_pressure[0] = pressure * (1.0 / 128.0);
696         channelpressure = pressure;
697         for (int i = 0; i < voices.length; i++) {
698             if (voices[i].active)
699                 voices[i].setChannelPressure(pressure);
700         }
701     }
702 }
703
704 public int getChannelPressure() {
705     synchronized (control_mutex) {
706         return channelpressure;
707     }
708 }
709
710 protected void applyInstrumentCustomization() {
711     if (cds_control_connections == null
712         && cds_channelpressure_connections == null
713         && cds_polypressure_connections == null) {
714         return;
715     }
716
717     ModelInstrument src_instrument = current_instrument.getSourceInstrument();
718     ModelPerformer[] performers = src_instrument.getPerformers();
719     ModelPerformer[] new_performers = new ModelPerformer[performers.length];
720     for (int i = 0; i < new_performers.length; i++) {
721         ModelPerformer performer = performers[i];
722         ModelPerformer new_performer = new ModelPerformer();
723         new_performer.setName(performer.getName());
724         new_performer.setExclusiveClass(performer.getExclusiveClass());
725         new_performer.setKeyFrom(performer.getKeyFrom());
726         new_performer.setKeyTo(performer.getKeyTo());
727         new_performer.setVelFrom(performer.getVelFrom());
728         new_performer.setVelTo(performer.getVelTo());
729         new_performer.getOscillators().addAll(performer.getOscillators());
730         new_performer.getConnectionBlocks().addAll(
731             performer.getConnectionBlocks());
732         new_performers[i] = new_performer;
733
734         List<ModelConnectionBlock> connblocks =
735             new_performer.getConnectionBlocks();
736
737         if (cds_control_connections != null) {
738             String cc = Integer.toString(cds_control_number);
739             Iterator<ModelConnectionBlock> iter = connblocks.iterator();
740             while (iter.hasNext()) {
741                 ModelConnectionBlock conn = iter.next();

```

```

742         ModelSource[] sources = conn.getSources();
743         boolean removeok = false;
744         if (sources != null) {
745             for (int j = 0; j < sources.length; j++) {
746                 ModelSource src = sources[j];
747                 if ("midi_cc".equals(src.getIdentifier().getObject())
748                     && cc.equals(src.getIdentifier().getVariable())) {
749                     removeok = true;
750                 }
751             }
752         }
753         if (removeok)
754             iter.remove();
755     }
756     for (int j = 0; j < cds_control_connections.length; j++)
757         connblocks.add(cds_control_connections[j]);
758 }
759
760 if (cds_polypressure_connections != null) {
761     Iterator<ModelConnectionBlock> iter = connblocks.iterator();
762     while (iter.hasNext()) {
763         ModelConnectionBlock conn = iter.next();
764         ModelSource[] sources = conn.getSources();
765         boolean removeok = false;
766         if (sources != null) {
767             for (int j = 0; j < sources.length; j++) {
768                 ModelSource src = sources[j];
769                 if ("midi".equals(src.getIdentifier().getObject())
770                     && "poly_pressure".equals(
771                         src.getIdentifier().getVariable())) {
772                     removeok = true;
773                 }
774             }
775         }
776         if (removeok)
777             iter.remove();
778     }
779     for (int j = 0; j < cds_polypressure_connections.length; j++)
780         connblocks.add(cds_polypressure_connections[j]);
781 }
782
783
784 if (cds_channelpressure_connections != null) {
785     Iterator<ModelConnectionBlock> iter = connblocks.iterator();
786     while (iter.hasNext()) {
787         ModelConnectionBlock conn = iter.next();
788         ModelSource[] sources = conn.getSources();
789         boolean removeok = false;
790         if (sources != null) {
791             for (int j = 0; j < sources.length; j++) {
792                 ModelIdentifier srcid = sources[j].getIdentifier();
793                 if ("midi".equals(srcid.getObject()) &&
794                     "channel_pressure".equals(srcid.getVariable())) {
795                     removeok = true;
796                 }
797             }
798         }
799         if (removeok)
800             iter.remove();
801     }
802     for (int j = 0; j < cds_channelpressure_connections.length; j++)
803         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         if (d == 1) {
844             double scale = (r / 64.0 - 1.0) * 9600.0;
845             ModelConnectionBlock conn;
846             if (scale > 0) {
847                 conn = new ModelConnectionBlock(
848                     new ModelSource(sid,
849                         ModelStandardTransform.DIRECTION_MAX2MIN,
850                         ModelStandardTransform.POLARITY_UNIPOLAR,
851                         ModelStandardTransform.TRANSFORM_LINEAR),
852                     -scale,
853                     new ModelDestination(
854                         ModelDestination.DESTINATION_FILTER_FREQ));
855             } else {
856                 conn = new ModelConnectionBlock(
857                     new ModelSource(sid,
858                         ModelStandardTransform.DIRECTION_MIN2MAX,
859                         ModelStandardTransform.POLARITY_UNIPOLAR,
860                         ModelStandardTransform.TRANSFORM_LINEAR),
861                     scale,
862                     new ModelDestination(
863                         ModelDestination.DESTINATION_FILTER_FREQ));
864             }
865         }
866     }

```

```

866         conns.add(conn);
867     }
868     if (d == 2) {
869         final double scale = (r / 64.0);
870         ModelTransform mt = new ModelTransform() {
871             double s = scale;
872             public double transform(double value) {
873                 if (s < 1)
874                     value = s + (value * (1.0 - s));
875                 else if (s > 1)
876                     value = 1 + (value * (s - 1.0));
877                 else
878                     return 0;
879                 return -((5.0 / 12.0) / Math.log(10)) * Math.log(value);
880             }
881         };
882
883         ModelConnectionBlock conn = new ModelConnectionBlock(
884             new ModelSource(sid, mt), -960,
885             new ModelDestination(ModelDestination.DESTINATION_GAIN));
886         conns.add(conn);
887     }
888
889     if (d == 3) {
890         double scale = (r / 64.0 - 1.0) * 9600.0;
891         ModelConnectionBlock conn = new ModelConnectionBlock(
892             new ModelSource(ModelSource.SOURCE_LF01,
893                 ModelStandardTransform.DIRECTION_MIN2MAX,
894                 ModelStandardTransform.POLARITY_BIPOLAR,
895                 ModelStandardTransform.TRANSFORM_LINEAR),
896             new ModelSource(sid,
897                 ModelStandardTransform.DIRECTION_MIN2MAX,
898                 ModelStandardTransform.POLARITY_UNIPOLAR,
899                 ModelStandardTransform.TRANSFORM_LINEAR),
900             scale,
901             new ModelDestination(
902                 ModelDestination.DESTINATION_PITCH));
903         conns.add(conn);
904     }
905
906     if (d == 4) {
907         double scale = (r / 128.0) * 2400.0;
908         ModelConnectionBlock conn = new ModelConnectionBlock(
909             new ModelSource(ModelSource.SOURCE_LF01,
910                 ModelStandardTransform.DIRECTION_MIN2MAX,
911                 ModelStandardTransform.POLARITY_BIPOLAR,
912                 ModelStandardTransform.TRANSFORM_LINEAR),
913             new ModelSource(sid,
914                 ModelStandardTransform.DIRECTION_MIN2MAX,
915                 ModelStandardTransform.POLARITY_UNIPOLAR,
916                 ModelStandardTransform.TRANSFORM_LINEAR),
917             scale,
918             new ModelDestination(
919                 ModelDestination.DESTINATION_FILTER_FREQ));
920         conns.add(conn);
921     }
922
923     if (d == 5) {
924         final double scale = (r / 127.0);
925
926         ModelTransform mt = new ModelTransform() {
927             double s = scale;
928             public double transform(double value) {
929                 return -((5.0 / 12.0) / Math.log(10))

```

```

928         * Math.log(1 - value * s);
929     }
930 };
931
932     ModelConnectionBlock conn = new ModelConnectionBlock(
933         new ModelSource(ModelSource.SOURCE_LF01,
934             ModelStandardTransform.DIRECTION_MIN2MAX,
935             ModelStandardTransform.POLARITY_UNIPOLAR,
936             ModelStandardTransform.TRANSFORM_LINEAR),
937         new ModelSource(sid, mt),
938         -960,
939         new ModelDestination(
940             ModelDestination.DESTINATION_GAIN));
941     conns.add(conn);
942 }
943 }
944
945     return conns.toArray(new ModelConnectionBlock[conns.size()]);
946 }
947
948 public void mapPolyPressureToDestination(int[] destination, int[] range) {
949     current_instrument = null;
950     if (destination.length == 0) {
951         cds_polypressure_connections = null;
952         return;
953     }
954     cds_polypressure_connections
955         = createModelConnections(
956             new ModelIdentifier("midi", "poly_pressure"),
957             destination, range);
958 }
959
960 public void mapChannelPressureToDestination(int[] destination, int[] range) {
961     current_instrument = null;
962     if (destination.length == 0) {
963         cds_channelpressure_connections = null;
964         return;
965     }
966     cds_channelpressure_connections
967         = createModelConnections(
968             new ModelIdentifier("midi", "channel_pressure"),
969             destination, range);
970 }
971
972 public void mapControlToDestination(int control, int[] destination, int[] range) {
973
974     if (!((control >= 0x01 && control <= 0x1F)
975         || (control >= 0x40 && control <= 0x5F))) {
976         cds_control_connections = null;
977         return;
978     }
979
980     current_instrument = null;
981     cds_control_number = control;
982     if (destination.length == 0) {
983         cds_control_connections = null;
984         return;
985     }
986     cds_control_connections
987         = createModelConnections(
988             new ModelIdentifier("midi_cc", Integer.toString(control)),
989             destination, range);

```



```

990     }
991
992     public void controlChangePerNote(int noteNumber, int controller, int value) {
993
994     /*
995     CC# | nn | Name | vv | default | description
996     -----|-----|-----|-----|-----|-----
997
998 7 | 07H | Note Volume | 00H-40H-7FH | 40H | 0-100-(127/64)*100(%) (
999   Relative)
1000 10 | 0AH | *Pan | 00H-7FH absolute | Preset Value | Left-Center-Right (absolute)
1001 33-63 | 21-3FH | LSB for | 01H-1FH | | |
1002 71 | 47H | Timbre/Harmonic Intensity | 00H-40H-7FH | 40H (???) |
1003 72 | 48H | Release Time | 00H-40H-7FH | 40H (???) |
1004 73 | 49H | Attack Time | 00H-40H-7FH | 40H (???) |
1005 74 | 4AH | Brightness | 00H-40H-7FH | 40H (???) |
1006 75 | 4BH | Decay Time | 00H-40H-7FH | 40H (???) |
1007 76 | 4CH | Vibrato Rate | 00H-40H-7FH | 40H (???) |
1008 77 | 4DH | Vibrato Depth | 00H-40H-7FH | 40H (???) |
1009 78 | 4EH | Vibrato Delay | 00H-40H-7FH | 40H (???) |
1010 91 | 5BH | *Reverb Send | 00H-7FH absolute | Preset Value | Left-Center-Right (absolute)
1011 93 | 5DH | *Chorus Send | 00H-7FH absolute | Preset Value | Left-Center-Right (absolute)
1012 120 | 78H | **Fine Tuning | 00H-40H-7FH | 40H (???) |
1013 121 | 79H | **Coarse Tuning | 00H-40H-7FH | 40H (???) |
1014 */
1015
1016     if (keybasedcontroller_active == null) {
1017         keybasedcontroller_active = new boolean[128][];
1018         keybasedcontroller_value = new double[128][];
1019     }
1020     if (keybasedcontroller_active[noteNumber] == null) {
1021         keybasedcontroller_active[noteNumber] = new boolean[128];
1022         Arrays.fill(keybasedcontroller_active[noteNumber], false);
1023         keybasedcontroller_value[noteNumber] = new double[128];
1024         Arrays.fill(keybasedcontroller_value[noteNumber], 0);
1025     }
1026
1027     if (value == -1) {
1028         keybasedcontroller_active[noteNumber][controller] = false;
1029     } else {
1030         keybasedcontroller_active[noteNumber][controller] = true;
1031         keybasedcontroller_value[noteNumber][controller] = value / 128.0;
1032     }
1033
1034     if (controller < 120) {
1035         for (int i = 0; i < voices.length; i++)
1036             if (voices[i].active)
1037                 voices[i].controlChange(controller, -1);
1038     } else if (controller == 120) {
1039         for (int i = 0; i < voices.length; i++)
1040             if (voices[i].active)
1041                 voices[i].rpnChange(1, -1);
1042     } else if (controller == 121) {
1043         for (int i = 0; i < voices.length; i++)
1044             if (voices[i].active)
1045                 voices[i].rpnChange(2, -1);
1046     }
1047
1048     }
1049
1050     public int getControlPerNote(int noteNumber, int controller) {
1051         if (keybasedcontroller_active == null)

```

```

1050         return -1;
1051     if (keybasedcontroller_active[noteNumber] == null)
1052         return -1;
1053     if (!keybasedcontroller_active[noteNumber][controller])
1054         return -1;
1055     return (int)(keybasedcontroller_value[noteNumber][controller] * 128);
1056 }
1057
1058 public void controlChange(int controller, int value) {
1059     controller = restrict7Bit(controller);
1060     value = restrict7Bit(value);
1061     if (current_mixer != null)
1062         current_mixer.controlChange(controller, value);
1063
1064     synchronized (control_mutex) {
1065         switch (controller) {
1066             /*
1067             Map<String, int[]>co_midi_rpn_rpn_i = new HashMap<String, int[]>();
1068             Map<String, double[]>co_midi_rpn_rpn = new HashMap<String, double[]>();
1069             Map<String, int[]>co_midi_nrpn_nrpn_i = new HashMap<String, int[]>();
1070             Map<String, double[]>co_midi_nrpn_nrpn = new HashMap<String, double[]>();
1071             */
1072
1073             case 5:
1074                 // This produce asin-like curve
1075                 // as described in General Midi Level 2 Specification, page 6
1076                 double x = -Math.asin((value / 128.0) * 2 - 1) / Math.PI + 0.5;
1077                 x = Math.pow(100000.0, x) / 100.0; // x is now cent/msec
1078                 // Convert x from cent/msec to key/controlbuffertime
1079                 x = x / 100.0; // x is now keys/msec
1080                 x = x * 1000.0; // x is now keys/sec
1081                 x = x / synthesizer.getControlRate(); // x is now keys/controlbuffertime
1082                 portamento_time = x;
1083                 break;
1084             case 6:
1085             case 38:
1086             case 96:
1087             case 97:
1088                 int val = 0;
1089                 if (nrpn_control != RPN_NULL_VALUE) {
1090                     int[] val_i = co_midi_nrpn_nrpn_i.get(nrpn_control);
1091                     if (val_i != null)
1092                         val = val_i[0];
1093                 }
1094                 if (rpn_control != RPN_NULL_VALUE) {
1095                     int[] val_i = co_midi_rpn_rpn_i.get(rpn_control);
1096                     if (val_i != null)
1097                         val = val_i[0];
1098                 }
1099
1100                 if (controller == 6)
1101                     val = (val & 127) + (value << 7);
1102                 else if (controller == 38)
1103                     val = (val & (127 << 7)) + value;
1104                 else if (controller == 96 || controller == 97) {
1105                     int step = 1;
1106                     if (rpn_control == 2 || rpn_control == 3 || rpn_control == 4)
1107                         step = 128;
1108                     if (controller == 96)
1109                         val += step;
1110                     if (controller == 97)
1111                         val -= step;

```

```

1112     }
1113
1114     if (nrpn_control != RPN_NULL_VALUE)
1115         nrpnChange(nrpn_control, val);
1116     if (rpn_control != RPN_NULL_VALUE)
1117         rpnChange(rpn_control, val);
1118
1119     break;
1120 case 64: // Hold1 (Damper) (cc#64)
1121     boolean on = value >= 64;
1122     if (sustain != on) {
1123         sustain = on;
1124         if (!on) {
1125             for (int i = 0; i < voices.length; i++) {
1126                 if (voices[i].active && voices[i].sustain &&
1127                     voices[i].channel == channel) {
1128                     voices[i].sustain = false;
1129                     if (!voices[i].on) {
1130                         voices[i].on = true;
1131                         voices[i].noteOff(0);
1132                     }
1133                 }
1134             }
1135         } else {
1136             for (int i = 0; i < voices.length; i++)
1137                 if (voices[i].active && voices[i].channel == channel)
1138                     voices[i].redamp();
1139         }
1140     }
1141     break;
1142 case 65:
1143     //allNotesOff();
1144     portamento = value >= 64;
1145     portamento_lastnote[0] = -1;
1146     /*
1147     for (int i = 0; i < portamento_lastnote.length; i++)
1148         portamento_lastnote[i] = -1;
1149     */
1150     portamento_lastnote_ix = 0;
1151     break;
1152 case 66: // Sostenuto (cc#66)
1153     on = value >= 64;
1154     if (on) {
1155         for (int i = 0; i < voices.length; i++) {
1156             if (voices[i].active && voices[i].on &&
1157                 voices[i].channel == channel) {
1158                 voices[i].sostenuto = true;
1159             }
1160         }
1161     }
1162     if (!on) {
1163         for (int i = 0; i < voices.length; i++) {
1164             if (voices[i].active && voices[i].sostenuto &&
1165                 voices[i].channel == channel) {
1166                 voices[i].sostenuto = false;
1167                 if (!voices[i].on) {
1168                     voices[i].on = true;
1169                     voices[i].noteOff(0);
1170                 }
1171             }
1172         }
1173     }

```

```

1174         break;
1175     case 98:
1176         nrpn_control = (nrpn_control & (127 << 7)) + value;
1177         rpn_control = RPN_NULL_VALUE;
1178         break;
1179     case 99:
1180         nrpn_control = (nrpn_control & 127) + (value << 7);
1181         rpn_control = RPN_NULL_VALUE;
1182         break;
1183     case 100:
1184         rpn_control = (rpn_control & (127 << 7)) + value;
1185         nrpn_control = RPN_NULL_VALUE;
1186         break;
1187     case 101:
1188         rpn_control = (rpn_control & 127) + (value << 7);
1189         nrpn_control = RPN_NULL_VALUE;
1190         break;
1191     case 120:
1192         allSoundOff();
1193         break;
1194     case 121:
1195         resetAllControllers(value == 127);
1196         break;
1197     case 122:
1198         localControl(value >= 64);
1199         break;
1200     case 123:
1201         allNotesOff();
1202         break;
1203     case 124:
1204         setOmni(false);
1205         break;
1206     case 125:
1207         setOmni(true);
1208         break;
1209     case 126:
1210         if (value == 1)
1211             setMono(true);
1212         break;
1213     case 127:
1214         setMono(false);
1215         break;
1216
1217     default:
1218         break;
1219 }
1220
1221 co_midi_cc_cc[controller][0] = value * (1.0 / 128.0);
1222
1223 if (controller == 0x00) {
1224     bank = /*(bank & 127) +*/ (value << 7);
1225     return;
1226 }
1227
1228 if (controller == 0x20) {
1229     bank = (bank & (127 << 7)) + value;
1230     return;
1231 }
1232
1233 this.controller[controller] = value;
1234 if(controller < 0x20)
1235     this.controller[controller + 0x20] = 0;

```

```

1236         for (int i = 0; i < voices.length; i++)
1237             if (voices[i].active)
1238                 voices[i].controlChange(controller, value);
1239     }
1240 }
1241
1242
1243
1244 public int getController(int controller) {
1245     synchronized (control_mutex) {
1246         // Should only return lower 7 bits,
1247         // even when controller is "boosted" higher.
1248         return this.controller[controller] & 127;
1249     }
1250 }
1251
1252 public void tuningChange(int program) {
1253     tuningChange(0, program);
1254 }
1255
1256 public void tuningChange(int bank, int program) {
1257     synchronized (control_mutex) {
1258         tuning = synthesizer.getTuning(new Patch(bank, program));
1259     }
1260 }
1261
1262 public void programChange(int program) {
1263     programChange(bank, program);
1264 }
1265
1266 public void programChange(int bank, int program) {
1267     bank = restrict14Bit(bank);
1268     program = restrict7Bit(program);
1269     synchronized (control_mutex) {
1270         mainmixer.activity();
1271         if(this.bank != bank || this.program != program)
1272         {
1273             this.bank = bank;
1274             this.program = program;
1275             current_instrument = null;
1276         }
1277     }
1278 }
1279
1280 public int getProgram() {
1281     synchronized (control_mutex) {
1282         return program;
1283     }
1284 }
1285
1286 public void setPitchBend(int bend) {
1287     bend = restrict14Bit(bend);
1288     if (current_mixer != null)
1289         current_mixer.setPitchBend(bend);
1290     synchronized (control_mutex) {
1291         mainmixer.activity();
1292         co_midi_pitch[0] = bend * (1.0 / 16384.0);
1293         pitchbend = bend;
1294         for (int i = 0; i < voices.length; i++)
1295             if (voices[i].active)
1296                 voices[i].setPitchBend(bend);
1297     }

```

```

1298     }
1299
1300     public int getPitchBend() {
1301         synchronized (control_mutex) {
1302             return pitchbend;
1303         }
1304     }
1305
1306     public void nrpnChange(int controller, int value) {
1307
1308         /*
1309         System.out.println("(" + channel + ").nrpnChange("
1310             + Integer.toHexString(controller >> 7)
1311             + " " + Integer.toHexString(controller & 127)
1312             + ", " + Integer.toHexString(value >> 7)
1313             + " " + Integer.toHexString(value & 127) + ")");
1314         */
1315
1316         if (synthesizer.getGeneralMidiMode() == 0) {
1317             if (controller == (0x01 << 7) + (0x08)) // Vibrato Rate
1318                 controlChange(76, value >> 7);
1319             if (controller == (0x01 << 7) + (0x09)) // Vibrato Depth
1320                 controlChange(77, value >> 7);
1321             if (controller == (0x01 << 7) + (0x0A)) // Vibrato Delay
1322                 controlChange(78, value >> 7);
1323             if (controller == (0x01 << 7) + (0x20)) // Brightness
1324                 controlChange(74, value >> 7);
1325             if (controller == (0x01 << 7) + (0x21)) // Filter Resonance
1326                 controlChange(71, value >> 7);
1327             if (controller == (0x01 << 7) + (0x63)) // Attack Time
1328                 controlChange(73, value >> 7);
1329             if (controller == (0x01 << 7) + (0x64)) // Decay Time
1330                 controlChange(75, value >> 7);
1331             if (controller == (0x01 << 7) + (0x66)) // Release Time
1332                 controlChange(72, value >> 7);
1333
1334             if (controller >> 7 == 0x18) // Pitch coarse
1335                 controlChangePerNote(controller % 128, 120, value >> 7);
1336             if (controller >> 7 == 0x1A) // Volume
1337                 controlChangePerNote(controller % 128, 7, value >> 7);
1338             if (controller >> 7 == 0x1C) // Panpot
1339                 controlChangePerNote(controller % 128, 10, value >> 7);
1340             if (controller >> 7 == 0x1D) // Reverb
1341                 controlChangePerNote(controller % 128, 91, value >> 7);
1342             if (controller >> 7 == 0x1E) // Chorus
1343                 controlChangePerNote(controller % 128, 93, value >> 7);
1344         }
1345
1346         int[] val_i = co_midi_nrpn_nrpn_i.get(controller);
1347         double[] val_d = co_midi_nrpn_nrpn.get(controller);
1348         if (val_i == null) {
1349             val_i = new int[1];
1350             co_midi_nrpn_nrpn_i.put(controller, val_i);
1351         }
1352         if (val_d == null) {
1353             val_d = new double[1];
1354             co_midi_nrpn_nrpn.put(controller, val_d);
1355         }
1356         val_i[0] = value;
1357         val_d[0] = val_i[0] * (1.0 / 16384.0);
1358
1359         for (int i = 0; i < voices.length; i++)

```

```

1360         if (voices[i].active)
1361             voices[i].nrpnChange(controller, val_i[0]);
1362
1363     }
1364
1365     public void rpnChange(int controller, int value) {
1366
1367         /*
1368         System.out.println("(" + channel + ").rpnChange("
1369             + Integer.toHexString(controller >> 7)
1370             + " " + Integer.toHexString(controller & 127)
1371             + ", " + Integer.toHexString(value >> 7)
1372             + " " + Integer.toHexString(value & 127) + ")");
1373         */
1374
1375         if (controller == 3) {
1376             tuning_program = (value >> 7) & 127;
1377             tuningChange(tuning_bank, tuning_program);
1378         }
1379         if (controller == 4) {
1380             tuning_bank = (value >> 7) & 127;
1381         }
1382
1383         int[] val_i = co_midi_rpn_rpn_i.get(controller);
1384         double[] val_d = co_midi_rpn_rpn.get(controller);
1385         if (val_i == null) {
1386             val_i = new int[1];
1387             co_midi_rpn_rpn_i.put(controller, val_i);
1388         }
1389         if (val_d == null) {
1390             val_d = new double[1];
1391             co_midi_rpn_rpn.put(controller, val_d);
1392         }
1393         val_i[0] = value;
1394         val_d[0] = val_i[0] * (1.0 / 16384.0);
1395
1396         for (int i = 0; i < voices.length; i++)
1397             if (voices[i].active)
1398                 voices[i].rpnChange(controller, val_i[0]);
1399     }
1400
1401     public void resetAllControllers() {
1402         resetAllControllers(false);
1403     }
1404
1405     public void resetAllControllers(boolean allControls) {
1406         synchronized (control_mutex) {
1407             mainmixer.activity();
1408
1409             for (int i = 0; i < 128; i++) {
1410                 setPolyPressure(i, 0);
1411             }
1412             setChannelPressure(0);
1413             setPitchBend(8192);
1414             for (int i = 0; i < 128; i++) {
1415                 if (!dontResetControls[i])
1416                     controlChange(i, 0);
1417             }
1418
1419             controlChange(71, 64); // Filter Resonance
1420             controlChange(72, 64); // Release Time
1421             controlChange(73, 64); // Attack Time

```

```

1422     controlChange(74, 64); // Brightness
1423     controlChange(75, 64); // Decay Time
1424     controlChange(76, 64); // Vibrato Rate
1425     controlChange(77, 64); // Vibrato Depth
1426     controlChange(78, 64); // Vibrato Delay
1427
1428     controlChange(8, 64); // Balance
1429     controlChange(11, 127); // Expression
1430     controlChange(98, 127); // NRPN Null
1431     controlChange(99, 127); // NRPN Null
1432     controlChange(100, 127); // RPN = Null
1433     controlChange(101, 127); // RPN = Null
1434
1435     // see DLS 2.1 (Power-on Default Values)
1436     if (allControls) {
1437
1438         keybasedcontroller_active = null;
1439         keybasedcontroller_value = null;
1440
1441         controlChange(7, 100); // Volume
1442         controlChange(10, 64); // Pan
1443         controlChange(91, 40); // Reverb
1444
1445         for (int controller : co_midi_rpn_rpn.keySet()) {
1446             // don't reset tuning settings
1447             if (controller != 3 && controller != 4)
1448                 rpnChange(controller, 0);
1449         }
1450         for (int controller : co_midi_nrpn_nrpn.keySet())
1451             nrpnChange(controller, 0);
1452         rpnChange(0, 2 << 7); // Bitch Bend sensitivity
1453         rpnChange(1, 64 << 7); // Channel fine tuning
1454         rpnChange(2, 64 << 7); // Channel Coarse Tuning
1455         rpnChange(5, 64); // Modulation Depth, +/- 50 cent
1456
1457         tuning_bank = 0;
1458         tuning_program = 0;
1459         tuning = new SoftTuning();
1460     }
1461 }
1462
1463 }
1464
1465 }
1466
1467 public void allNotesOff() {
1468     if (current_mixer != null)
1469         current_mixer.allNotesOff();
1470     synchronized (control_mutex) {
1471         for (int i = 0; i < voices.length; i++)
1472             if (voices[i].on && voices[i].channel == channel
1473                 && voices[i].releaseTriggered == false) {
1474                 voices[i].noteOff(0);
1475             }
1476     }
1477 }
1478
1479 public void allSoundOff() {
1480     if (current_mixer != null)
1481         current_mixer.allSoundOff();
1482     synchronized (control_mutex) {
1483         for (int i = 0; i < voices.length; i++)
1484             if (voices[i].on && voices[i].channel == channel)

```



```

1484         voices[i].soundOff();
1485     }
1486 }
1487
1488 public boolean localControl(boolean on) {
1489     return false;
1490 }
1491
1492 public void setMono(boolean on) {
1493     if (current_mixer != null)
1494         current_mixer.setMono(on);
1495     synchronized (control_mutex) {
1496         allNotesOff();
1497         mono = on;
1498     }
1499 }
1500
1501 public boolean getMono() {
1502     synchronized (control_mutex) {
1503         return mono;
1504     }
1505 }
1506
1507 public void setOmni(boolean on) {
1508     if (current_mixer != null)
1509         current_mixer.setOmni(on);
1510     allNotesOff();
1511     // Omni is not supported by GM2
1512 }
1513
1514 public boolean getOmni() {
1515     return false;
1516 }
1517
1518 public void setMute(boolean mute) {
1519     if (current_mixer != null)
1520         current_mixer.setMute(mute);
1521     synchronized (control_mutex) {
1522         this.mute = mute;
1523         for (int i = 0; i < voices.length; i++)
1524             if (voices[i].active && voices[i].channel == channel)
1525                 voices[i].setMute(mute);
1526     }
1527 }
1528
1529 public boolean getMute() {
1530     synchronized (control_mutex) {
1531         return mute;
1532     }
1533 }
1534
1535 public void setSolo(boolean soloState) {
1536     if (current_mixer != null)
1537         current_mixer.setSolo(soloState);
1538
1539     synchronized (control_mutex) {
1540         this.solo = soloState;
1541
1542         boolean soloinuse = false;
1543         for (SoftChannel c : synthesizer.channels) {
1544             if (c.solo) {
1545                 soloinuse = true;

```

```

1546         break;
1547     }
1548 }
1549
1550 if (!soloinuse) {
1551     for (SoftChannel c : synthesizer.channels)
1552         c.setSoloMute(false);
1553     return;
1554 }
1555
1556 for (SoftChannel c : synthesizer.channels)
1557     c.setSoloMute(!c.solo);
1558
1559 }
1560
1561 }
1562
1563 private void setSoloMute(boolean mute) {
1564     synchronized (control_mutex) {
1565         if (solomute == mute)
1566             return;
1567         this.solomute = mute;
1568         for (int i = 0; i < voices.length; i++)
1569             if (voices[i].active && voices[i].channel == channel)
1570                 voices[i].setSoloMute(solomute);
1571     }
1572 }
1573
1574 public boolean getSolo() {
1575     synchronized (control_mutex) {
1576         return solo;
1577     }
1578 }
1579 }

```

82 com/sun/media/sound/SoftChannelProxy.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 javax.sound.midi.MidiChannel;
28
29 /**
30 * A MidiChannel proxy object used for external access to synthesizer internal
31 * channel objects.
32 *
33 * @author Karl Helgason
34 */
35 public class SoftChannelProxy implements MidiChannel {
36
37     private MidiChannel channel = null;
38
39     public MidiChannel getChannel() {
40         return channel;
41     }
42
43     public void setChannel(MidiChannel channel) {
44         this.channel = channel;
45     }
46
47     public void allNotesOff() {
48         if (channel == null)
49             return;
50         channel.allNotesOff();
51     }
52
53     public void allSoundOff() {
54         if (channel == null)
55             return;
56         channel.allSoundOff();
57     }
58
59     public void controlChange(int controller, int value) {
60         if (channel == null)
```

```

61         return;
62         channel.controlChange(controller, value);
63     }
64
65     public int getChannelPressure() {
66         if (channel == null)
67             return 0;
68         return channel.getChannelPressure();
69     }
70
71     public int getController(int controller) {
72         if (channel == null)
73             return 0;
74         return channel.getController(controller);
75     }
76
77     public boolean getMono() {
78         if (channel == null)
79             return false;
80         return channel.getMono();
81     }
82
83     public boolean getMute() {
84         if (channel == null)
85             return false;
86         return channel.getMute();
87     }
88
89     public boolean getOmni() {
90         if (channel == null)
91             return false;
92         return channel.getOmni();
93     }
94
95     public int getPitchBend() {
96         if (channel == null)
97             return 8192;
98         return channel.getPitchBend();
99     }
100
101     public int getPolyPressure(int noteNumber) {
102         if (channel == null)
103             return 0;
104         return channel.getPolyPressure(noteNumber);
105     }
106
107     public int getProgram() {
108         if (channel == null)
109             return 0;
110         return channel.getProgram();
111     }
112
113     public boolean getSolo() {
114         if (channel == null)
115             return false;
116         return channel.getSolo();
117     }
118
119     public boolean localControl(boolean on) {
120         if (channel == null)
121             return false;
122         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

```
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.Arrays;
28
29 /**
30 * A chorus effect made using LFO and variable delay. One for each channel
31 * (left,right), with different starting phase for stereo effect.
32 *
33 * @author Karl Helgason
34 */
35 public class SoftChorus implements SoftAudioProcessor {
36
37     private static class VariableDelay {
38
39         private float[] delaybuffer;
40         private int rovepos = 0;
41         private float gain = 1;
42         private float rgain = 0;
43         private float delay = 0;
44         private float lastdelay = 0;
45         private float feedback = 0;
46
47         public VariableDelay(int maxbuffersize) {
48             delaybuffer = new float[maxbuffersize];
49         }
50
51         public void setDelay(float delay) {
52             this.delay = delay;
53         }
54
55         public void setFeedBack(float feedback) {
56             this.feedback = feedback;
57         }
58
59         public void setGain(float gain) {
60             this.gain = gain;
```

```

61     }
62
63     public void setReverbSendGain(float rgain) {
64         this.rgain = rgain;
65     }
66
67     public void processMix(float[] in, float[] out, float[] rout) {
68         float gain = this.gain;
69         float delay = this.delay;
70         float feedback = this.feedback;
71
72         float[] delaybuffer = this.delaybuffer;
73         int len = in.length;
74         float delaydelta = (delay - lastdelay) / len;
75         int rnlen = delaybuffer.length;
76         int rovepos = this.rovepos;
77
78         if (rout == null)
79             for (int i = 0; i < len; i++) {
80                 float r = rovepos - (lastdelay + 2) + rnlen;
81                 int ri = (int) r;
82                 float s = r - ri;
83                 float a = delaybuffer[ri % rnlen];
84                 float b = delaybuffer[(ri + 1) % rnlen];
85                 float o = a * (1 - s) + b * (s);
86                 out[i] += o * gain;
87                 delaybuffer[rovepos] = in[i] + o * feedback;
88                 rovepos = (rovepos + 1) % rnlen;
89                 lastdelay += delaydelta;
90             }
91         else
92             for (int i = 0; i < len; i++) {
93                 float r = rovepos - (lastdelay + 2) + rnlen;
94                 int ri = (int) r;
95                 float s = r - ri;
96                 float a = delaybuffer[ri % rnlen];
97                 float b = delaybuffer[(ri + 1) % rnlen];
98                 float o = a * (1 - s) + b * (s);
99                 out[i] += o * gain;
100                rout[i] += o * rgain;
101                delaybuffer[rovepos] = in[i] + o * feedback;
102                rovepos = (rovepos + 1) % rnlen;
103                lastdelay += delaydelta;
104            }
105         this.rovepos = rovepos;
106         lastdelay = delay;
107     }
108
109     public void processReplace(float[] in, float[] out, float[] rout) {
110         Arrays.fill(out, 0);
111         Arrays.fill(rout, 0);
112         processMix(in, out, rout);
113     }
114 }
115
116 private static class LFODelay {
117
118     private double phase = 1;
119     private double phase_step = 0;
120     private double depth = 0;
121     private VariableDelay vdelay;
122     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;

```

```

185 private double dirty_vdelay1L_depth;
186 private double dirty_vdelay1R_depth;
187 private float dirty_vdelay1L_feedback;
188 private float dirty_vdelay1R_feedback;
189 private float dirty_vdelay1L_reverbSendGain;
190 private float dirty_vdelay1R_reverbSendGain;
191 private float controlrate;
192
193 public void init(float samplerate, float controlrate) {
194     this.controlrate = controlrate;
195     vdelay1L = new LF0Delay(samplerate, controlrate);
196     vdelay1R = new LF0Delay(samplerate, controlrate);
197     vdelay1L.setGain(1.0f); // %
198     vdelay1R.setGain(1.0f); // %
199     vdelay1L.setPhase(0.5 * Math.PI);
200     vdelay1R.setPhase(0);
201
202     globalParameterControlChange(new int[]{0x01 * 128 + 0x02}, 0, 2);
203 }
204
205 public void globalParameterControlChange(int[] slothpath, long param,
206     long value) {
207     if (slothpath.length == 1) {
208         if (slothpath[0] == 0x01 * 128 + 0x02) {
209             if (param == 0) { // Chorus Type
210                 switch ((int)value) {
211                     case 0: // Chorus 1 0 (0%) 3 (0.4Hz) 5 (1.9ms) 0 (0%)
212                         globalParameterControlChange(slothpath, 3, 0);
213                         globalParameterControlChange(slothpath, 1, 3);
214                         globalParameterControlChange(slothpath, 2, 5);
215                         globalParameterControlChange(slothpath, 4, 0);
216                         break;
217                     case 1: // Chorus 2 5 (4%) 9 (1.1Hz) 19 (6.3ms) 0 (0%)
218                         globalParameterControlChange(slothpath, 3, 5);
219                         globalParameterControlChange(slothpath, 1, 9);
220                         globalParameterControlChange(slothpath, 2, 19);
221                         globalParameterControlChange(slothpath, 4, 0);
222                         break;
223                     case 2: // Chorus 3 8 (6%) 3 (0.4Hz) 19 (6.3ms) 0 (0%)
224                         globalParameterControlChange(slothpath, 3, 8);
225                         globalParameterControlChange(slothpath, 1, 3);
226                         globalParameterControlChange(slothpath, 2, 19);
227                         globalParameterControlChange(slothpath, 4, 0);
228                         break;
229                     case 3: // Chorus 4 16 (12%) 9 (1.1Hz) 16 (5.3ms) 0 (0%)
230                         globalParameterControlChange(slothpath, 3, 16);
231                         globalParameterControlChange(slothpath, 1, 9);
232                         globalParameterControlChange(slothpath, 2, 16);
233                         globalParameterControlChange(slothpath, 4, 0);
234                         break;
235                     case 4: // FB Chorus 64 (49%) 2 (0.2Hz) 24 (7.8ms) 0 (0%)
236                         globalParameterControlChange(slothpath, 3, 64);
237                         globalParameterControlChange(slothpath, 1, 2);
238                         globalParameterControlChange(slothpath, 2, 24);
239                         globalParameterControlChange(slothpath, 4, 0);
240                         break;
241                     case 5: // Flanger 112 (86%) 1 (0.1Hz) 5 (1.9ms) 0 (0%)
242                         globalParameterControlChange(slothpath, 3, 112);
243                         globalParameterControlChange(slothpath, 1, 1);
244                         globalParameterControlChange(slothpath, 2, 5);
245                         globalParameterControlChange(slothpath, 4, 0);
246                         break;

```

```

247         default:
248             break;
249     }
250 } else if (param == 1) { // Mod Rate
251     dirty_vdelay1L_rate = (value * 0.122);
252     dirty_vdelay1R_rate = (value * 0.122);
253     dirty = true;
254 } else if (param == 2) { // Mod Depth
255     dirty_vdelay1L_depth = ((value + 1) / 3200.0);
256     dirty_vdelay1R_depth = ((value + 1) / 3200.0);
257     dirty = true;
258 } else if (param == 3) { // Feedback
259     dirty_vdelay1L_feedback = (value * 0.00763f);
260     dirty_vdelay1R_feedback = (value * 0.00763f);
261     dirty = true;
262 }
263 if (param == 4) { // Send to Reverb
264     rgain = value * 0.00787f;
265     dirty_vdelay1L_reverbSendgain = (value * 0.00787f);
266     dirty_vdelay1R_reverbSendgain = (value * 0.00787f);
267     dirty = true;
268 }
269
270 }
271
272 }
273
274 public void processControlLogic() {
275     if (dirty) {
276         dirty = false;
277         vdelay1L.setRate(dirty_vdelay1L_rate);
278         vdelay1R.setRate(dirty_vdelay1R_rate);
279         vdelay1L.setDepth(dirty_vdelay1L_depth);
280         vdelay1R.setDepth(dirty_vdelay1R_depth);
281         vdelay1L.setFeedBack(dirty_vdelay1L_feedback);
282         vdelay1R.setFeedBack(dirty_vdelay1R_feedback);
283         vdelay1L.setReverbSendGain(dirty_vdelay1L_reverbSendgain);
284         vdelay1R.setReverbSendGain(dirty_vdelay1R_reverbSendgain);
285     }
286 }
287 double silentcounter = 1000;
288
289 public void processAudio() {
290
291     if (inputA.isSilent()) {
292         silentcounter += 1 / controlrate;
293
294         if (silentcounter > 1) {
295             if (!mix) {
296                 left.clear();
297                 right.clear();
298             }
299             return;
300         }
301     } else
302         silentcounter = 0;
303
304     float[] inputA = this.inputA.array();
305     float[] left = this.left.array();
306     float[] right = this.right == null ? null : this.right.array();
307     float[] reverb = rgain != 0 ? this.reverb.array() : null;
308

```

```

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

```
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 /**
28 * <code>SoftControl</code> are the basic controls
29 * used for control-rate processing.
30 *
31 * @author Karl Helgason
32 */
33 public interface SoftControl {
34
35     public double[] get(int instance, String name);
36 }
```

85 com/sun/media/sound/SoftCubicResampler.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 /**
28 * A resampler that uses third-order (cubic) interpolation.
29 *
30 * @author Karl Helgason
31 */
32 public class SoftCubicResampler extends SoftAbstractResampler {
33
34     public int getPadding() {
35         return 3;
36     }
37
38     public void interpolate(float[] in, float[] in_offset, float in_end,
39         float[] startpitch, float pitchstep, float[] out, int[] out_offset,
40         int out_end) {
41         float pitch = startpitch[0];
42         float ix = in_offset[0];
43         int ox = out_offset[0];
44         float ix_end = in_end;
45         int ox_end = out_end;
46         if (pitchstep == 0) {
47             while (ix < ix_end && ox < ox_end) {
48                 int iix = (int) ix;
49                 float fix = ix - iix;
50                 float y0 = in[iix - 1];
51                 float y1 = in[iix];
52                 float y2 = in[iix + 1];
53                 float y3 = in[iix + 2];
54                 float a0 = y3 - y2 + y1 - y0;
55                 float a1 = y0 - y1 - a0;
56                 float a2 = y2 - y0;
57                 float a3 = y1;
58                 //float fix2 = fix * fix;
59                 //out[ox++] = (a0 * fix + a1) * fix2 + (a2 * fix + a3);
60                 out[ox++] = ((a0 * fix + a1) * fix + a2) * fix + a3;
```

```

61         ix += pitch;
62     }
63 } else {
64     while (ix < ix_end && ox < ox_end) {
65         int iix = (int) ix;
66         float fix = ix - iix;
67         float y0 = in[iix - 1];
68         float y1 = in[iix];
69         float y2 = in[iix + 1];
70         float y3 = in[iix + 2];
71         float a0 = y3 - y2 + y1 - y0;
72         float a1 = y0 - y1 - a0;
73         float a2 = y2 - y0;
74         float a3 = y1;
75         //float fix2 = fix * fix;
76         //out[ox++] = (a0 * fix + a1) * fix2 + (a2 * fix + a3);
77         out[ox++] = ((a0 * fix + a1) * fix + a2) * fix + a3;
78         ix += pitch;
79         pitch += pitchstep;
80     }
81 }
82 in_offset[0] = ix;
83 out_offset[0] = ox;
84 startpitch[0] = pitch;
85
86 }
87 }

```

86 com/sun/media/sound/SoftEnvelopeGenerator.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 /**
28  * AHDSR control signal envelope generator.
29  *
30  * @author Karl Helgason
31  */
32 public class SoftEnvelopeGenerator implements SoftProcess {
33
34     public final static int EG_OFF = 0;
35     public final static int EG_DELAY = 1;
36     public final static int EG_ATTACK = 2;
37     public final static int EG_HOLD = 3;
38     public final static int EG_DECAY = 4;
39     public final static int EG_SUSTAIN = 5;
40     public final static int EG_RELEASE = 6;
41     public final static int EG_SHUTDOWN = 7;
42     public final static int EG_END = 8;
43     int max_count = 10;
44     int used_count = 0;
45     private int[] stage = new int[max_count];
46     private int[] stage_ix = new int[max_count];
47     private double[] stage_v = new double[max_count];
48     private int[] stage_count = new int[max_count];
49     private double[][] on = new double[max_count][1];
50     private double[][] active = new double[max_count][1];
51     private double[][] out = new double[max_count][1];
52     private double[][] delay = new double[max_count][1];
53     private double[][] attack = new double[max_count][1];
54     private double[][] hold = new double[max_count][1];
55     private double[][] decay = new double[max_count][1];
56     private double[][] sustain = new double[max_count][1];
57     private double[][] release = new double[max_count][1];
58     private double[][] shutdown = new double[max_count][1];
59     private double[][] release2 = new double[max_count][1];
60     private double[][] attack2 = new double[max_count][1];
```



```

61 private double[][] decay2 = new double[max_count][1];
62 private double control_time = 0;
63
64 public void reset() {
65     for (int i = 0; i < used_count; i++) {
66         stage[i] = 0;
67         on[i][0] = 0;
68         out[i][0] = 0;
69         delay[i][0] = 0;
70         attack[i][0] = 0;
71         hold[i][0] = 0;
72         decay[i][0] = 0;
73         sustain[i][0] = 0;
74         release[i][0] = 0;
75         shutdown[i][0] = 0;
76         attack2[i][0] = 0;
77         decay2[i][0] = 0;
78         release2[i][0] = 0;
79     }
80     used_count = 0;
81 }
82
83 public void init(SoftSynthesizer synth) {
84     control_time = 1.0 / synth.getControlRate();
85     processControlLogic();
86 }
87
88 public double[] get(int instance, String name) {
89     if (instance >= used_count)
90         used_count = instance + 1;
91     if (name == null)
92         return out[instance];
93     if (name.equals("on"))
94         return on[instance];
95     if (name.equals("active"))
96         return active[instance];
97     if (name.equals("delay"))
98         return delay[instance];
99     if (name.equals("attack"))
100         return attack[instance];
101     if (name.equals("hold"))
102         return hold[instance];
103     if (name.equals("decay"))
104         return decay[instance];
105     if (name.equals("sustain"))
106         return sustain[instance];
107     if (name.equals("release"))
108         return release[instance];
109     if (name.equals("shutdown"))
110         return shutdown[instance];
111     if (name.equals("attack2"))
112         return attack2[instance];
113     if (name.equals("decay2"))
114         return decay2[instance];
115     if (name.equals("release2"))
116         return release2[instance];
117
118     return null;
119 }
120
121 public void processControlLogic() {
122     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;

```

```

185         } else {
186             stage[i] = EG_ATTACK;
187             stage_count[i] = (int)(Math.pow(2,
188                 attack / 1200.0) / control_time);
189             stage_count[i] += (int)(attack2 / (control_time * 1000));
190             if (stage_count[i] < 0)
191                 stage_count[i] = 0;
192             stage_ix[i] = 0;
193         }
194     } else
195         stage_ix[i]--;
196     break;
197 case EG_ATTACK:
198     stage_ix[i]++;
199     if (stage_ix[i] >= stage_count[i]) {
200         out[i][0] = 1;
201         stage[i] = EG_HOLD;
202     } else {
203         // CONVEX attack
204         double a = ((double)stage_ix[i]) / ((double)stage_count[i]);
205         a = 1 + ((40.0 / 96.0) / Math.log(10)) * Math.log(a);
206         if (a < 0)
207             a = 0;
208         else if (a > 1)
209             a = 1;
210         out[i][0] = a;
211     }
212     break;
213 case EG_HOLD:
214     stage_ix[i]++;
215     if (stage_ix[i] >= stage_count[i]) {
216         stage[i] = EG_DECAY;
217         stage_count[i] = (int)(Math.pow(2,
218             this.decay[i][0] / 1200.0) / control_time);
219         stage_count[i] += (int)(this.decay2[i][0]/(control_time*1000));
220         if (stage_count[i] < 0)
221             stage_count[i] = 0;
222         stage_ix[i] = 0;
223     }
224     break;
225 case EG_DECAY:
226     stage_ix[i]++;
227     double sustain = this.sustain[i][0] * (1.0 / 1000.0);
228     if (stage_ix[i] >= stage_count[i]) {
229         out[i][0] = sustain;
230         stage[i] = EG_SUSTAIN;
231         if (sustain < 0.001) {
232             out[i][0] = 0;
233             active[i][0] = 0;
234             stage[i] = EG_END;
235         }
236     } else {
237         double m = ((double)stage_ix[i]) / ((double)stage_count[i]);
238         out[i][0] = (1 - m) + sustain * m;
239     }
240     break;
241 case EG_SUSTAIN:
242     break;
243 case EG_RELEASE:
244     stage_ix[i]++;
245     if (stage_ix[i] >= stage_count[i]) {
246         out[i][0] = 0;

```

```

247     active[i][0] = 0;
248     stage[i] = EG_END;
249 } else {
250     double m = ((double)stage_ix[i]) / ((double)stage_count[i]);
251     out[i][0] = (1 - m); // *stage_v[i];
252
253     if (on[i][0] < -0.5) {
254         stage_count[i] = (int)(Math.pow(2,
255             this.shutdown[i][0] / 1200.0) / control_time);
256         if (stage_count[i] < 0)
257             stage_count[i] = 0;
258         stage_v[i] = out[i][0];
259         stage_ix[i] = 0;
260         stage[i] = EG_SHUTDOWN;
261     }
262
263     // re-damping
264     if (on[i][0] > 0.5) {
265         sustain = this.sustain[i][0] * (1.0 / 1000.0);
266         if (out[i][0] > sustain) {
267             stage[i] = EG_DECAY;
268             stage_count[i] = (int)(Math.pow(2,
269                 this.decay[i][0] / 1200.0) / control_time);
270             stage_count[i] +=
271                 (int)(this.decay2[i][0]/(control_time*1000));
272             if (stage_count[i] < 0)
273                 stage_count[i] = 0;
274             m = (out[i][0] - 1) / (sustain - 1);
275             stage_ix[i] = (int) (stage_count[i] * m);
276         }
277     }
278
279     break;
280 case EG_SHUTDOWN:
281     stage_ix[i]++;
282     if (stage_ix[i] >= stage_count[i]) {
283         out[i][0] = 0;
284         active[i][0] = 0;
285         stage[i] = EG_END;
286     } else {
287         double m = ((double)stage_ix[i]) / ((double)stage_count[i]);
288         out[i][0] = (1 - m) * stage_v[i];
289     }
290     break;
291 default:
292     break;
293 }
294 }
295 }
296
297 }
298 }

```

87 com/sun/media/sound/SoftFilter.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 /**
28 * Infinite impulse response (IIR) filter class.
29 *
30 * The filters were implemented and adapted using algorithms from musicdsp.org
31 * archive: 1-RC and C filter, Simple 2-pole LP LP and HP filter, biquad,
32 * tweaked butterworth RBJ Audio-EQ-Cookbook, EQ filter cookbook
33 *
34 * @author Karl Helgason
35 */
36 public class SoftFilter {
37
38     public final static int FILTERTYPE_LP6 = 0x00;
39     public final static int FILTERTYPE_LP12 = 0x01;
40     public final static int FILTERTYPE_HP12 = 0x11;
41     public final static int FILTERTYPE_BP12 = 0x21;
42     public final static int FILTERTYPE_NP12 = 0x31;
43     public final static int FILTERTYPE_LP24 = 0x03;
44     public final static int FILTERTYPE_HP24 = 0x13;
45
46     //
47     // 0x0 = 1st-order, 6 dB/oct
48     // 0x1 = 2nd-order, 12 dB/oct
49     // 0x2 = 3rd-order, 18 dB/oct
50     // 0x3 = 4th-order, 24 dB/oct
51     //
52     // 0x00 = LP, Low Pass Filter
53     // 0x10 = HP, High Pass Filter
54     // 0x20 = BP, Band Pass Filter
55     // 0x30 = NP, Notch or Band Elimination Filter
56     //
57     private int filtertype = FILTERTYPE_LP6;
58     private float samplerate;
59     private float x1;
60     private float x2;
```

```

61     private float y1;
62     private float y2;
63     private float xx1;
64     private float xx2;
65     private float yy1;
66     private float yy2;
67     private float a0;
68     private float a1;
69     private float a2;
70     private float b1;
71     private float b2;
72     private float q;
73     private float gain = 1;
74     private float wet = 0;
75     private float last_wet = 0;
76     private float last_a0;
77     private float last_a1;
78     private float last_a2;
79     private float last_b1;
80     private float last_b2;
81     private float last_q;
82     private float last_gain;
83     private boolean last_set = false;
84     private double cutoff = 44100;
85     private double resonancedB = 0;
86     private boolean dirty = true;
87
88     public SoftFilter(float samplerate) {
89         this.samplerate = samplerate;
90         dirty = true;
91     }
92
93     public void setFrequency(double cent) {
94         if (cutoff == cent)
95             return;
96         cutoff = cent;
97         dirty = true;
98     }
99
100    public void setResonance(double db) {
101        if (resonancedB == db)
102            return;
103        resonancedB = db;
104        dirty = true;
105    }
106
107    public void reset() {
108        dirty = true;
109        last_set = false;
110        x1 = 0;
111        x2 = 0;
112        y1 = 0;
113        y2 = 0;
114        xx1 = 0;
115        xx2 = 0;
116        yy1 = 0;
117        yy2 = 0;
118        wet = 0.0f;
119        gain = 1.0f;
120        a0 = 0;
121        a1 = 0;
122        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;

```

```

185 float x2 = this.x2;
186 float y1 = this.y1;
187 float y2 = this.y2;
188 float xx1 = this.xx1;
189 float xx2 = this.xx2;
190 float yy1 = this.yy1;
191 float yy2 = this.yy2;
192
193 if (wet_delta != 0) {
194     for (int i = 0; i < len; i++) {
195         a0 += a0_delta;
196         a1 += a1_delta;
197         a2 += a2_delta;
198         b1 += b1_delta;
199         b2 += b2_delta;
200         gain += gain_delta;
201         wet += wet_delta;
202         float x = buffer[i];
203         float y = (a0*x + a1*x1 + a2*x2 - b1*y1 - b2*y2);
204         float xx = (y * gain) * wet + (x) * (1 - wet);
205         x2 = x1;
206         x1 = x;
207         y2 = y1;
208         y1 = y;
209         float yy = (a0*xx + a1*xx1 + a2*xx2 - b1*yy1 - b2*yy2);
210         buffer[i] = (yy * gain) * wet + (xx) * (1 - wet);
211         xx2 = xx1;
212         xx1 = xx;
213         yy2 = yy1;
214         yy1 = yy;
215     }
216 } else if (a0_delta == 0 && a1_delta == 0 && a2_delta == 0
217           && b1_delta == 0 && b2_delta == 0) {
218     for (int i = 0; i < len; i++) {
219         float x = buffer[i];
220         float y = (a0*x + a1*x1 + a2*x2 - b1*y1 - b2*y2);
221         float xx = (y * gain) * wet + (x) * (1 - wet);
222         x2 = x1;
223         x1 = x;
224         y2 = y1;
225         y1 = y;
226         float yy = (a0*xx + a1*xx1 + a2*xx2 - b1*yy1 - b2*yy2);
227         buffer[i] = (yy * gain) * wet + (xx) * (1 - wet);
228         xx2 = xx1;
229         xx1 = xx;
230         yy2 = yy1;
231         yy1 = yy;
232     }
233 } else {
234     for (int i = 0; i < len; i++) {
235         a0 += a0_delta;
236         a1 += a1_delta;
237         a2 += a2_delta;
238         b1 += b1_delta;
239         b2 += b2_delta;
240         gain += gain_delta;
241         float x = buffer[i];
242         float y = (a0*x + a1*x1 + a2*x2 - b1*y1 - b2*y2);
243         float xx = (y * gain) * wet + (x) * (1 - wet);
244         x2 = x1;
245         x1 = x;
246         y2 = y1;

```



```

247         y1 = y;
248         float yy = (a0*xx + a1*xx1 + a2*xx2 - b1*yy1 - b2*yy2);
249         buffer[i] = (yy * gain) * wet + (xx) * (1 - wet);
250         xx2 = xx1;
251         xx1 = xx;
252         yy2 = yy1;
253         yy1 = yy;
254     }
255 }
256
257 if (Math.abs(x1) < 1.0E-8)
258     x1 = 0;
259 if (Math.abs(x2) < 1.0E-8)
260     x2 = 0;
261 if (Math.abs(y1) < 1.0E-8)
262     y1 = 0;
263 if (Math.abs(y2) < 1.0E-8)
264     y2 = 0;
265 this.x1 = x1;
266 this.x2 = x2;
267 this.y1 = y1;
268 this.y2 = y2;
269 this.xx1 = xx1;
270 this.xx2 = xx2;
271 this.yy1 = yy1;
272 this.yy2 = yy2;
273 }
274
275 this.last_a0 = this.a0;
276 this.last_a1 = this.a1;
277 this.last_a2 = this.a2;
278 this.last_b1 = this.b1;
279 this.last_b2 = this.b2;
280 this.last_gain = this.gain;
281 this.last_wet = this.wet;
282
283 }
284
285 private double sinh(double x) {
286     return (Math.exp(x) - Math.exp(-x)) * 0.5;
287 }
288
289 public void filter2calc() {
290
291     double resonancedB = this.resonancedB;
292     if (resonancedB < 0)
293         resonancedB = 0;    // Negative dB are illegal.
294     if (resonancedB > 30)
295         resonancedB = 30;    // At least 22.5 dB is needed.
296     if (filtertype == FILTERTYPE_LP24 || filtertype == FILTERTYPE_HP24)
297         resonancedB *= 0.6;
298
299     if (filtertype == FILTERTYPE_BP12) {
300         wet = 1;
301         double r = (cutoff / samplerate);
302         if (r > 0.45)
303             r = 0.45;
304
305         double bandwidth = Math.PI * Math.pow(10.0, -(resonancedB / 20));
306
307         double omega = 2 * Math.PI * r;
308         double cs = Math.cos(omega);

```

```

309     double sn = Math.sin(omega);
310     double alpha = sn * sinh((Math.log(2)*bandwidth*omega) / (sn * 2));
311
312     double b0 = alpha;
313     double b1 = 0;
314     double b2 = -alpha;
315     double a0 = 1 + alpha;
316     double a1 = -2 * cs;
317     double a2 = 1 - alpha;
318
319     double cf = 1.0 / a0;
320     this.b1 = (float) (a1 * cf);
321     this.b2 = (float) (a2 * cf);
322     this.a0 = (float) (b0 * cf);
323     this.a1 = (float) (b1 * cf);
324     this.a2 = (float) (b2 * cf);
325 }
326
327 if (filtertype == FILTERTYPE_NP12) {
328     wet = 1;
329     double r = (cutoff / samplerate);
330     if (r > 0.45)
331         r = 0.45;
332
333     double bandwidth = Math.PI * Math.pow(10.0, -(resonancedB / 20));
334
335     double omega = 2 * Math.PI * r;
336     double cs = Math.cos(omega);
337     double sn = Math.sin(omega);
338     double alpha = sn * sinh((Math.log(2)*bandwidth*omega) / (sn*2));
339
340     double b0 = 1;
341     double b1 = -2 * cs;
342     double b2 = 1;
343     double a0 = 1 + alpha;
344     double a1 = -2 * cs;
345     double a2 = 1 - alpha;
346
347     double cf = 1.0 / a0;
348     this.b1 = (float)(a1 * cf);
349     this.b2 = (float)(a2 * cf);
350     this.a0 = (float)(b0 * cf);
351     this.a1 = (float)(b1 * cf);
352     this.a2 = (float)(b2 * cf);
353 }
354
355 if (filtertype == FILTERTYPE_LP12 || filtertype == FILTERTYPE_LP24) {
356     double r = (cutoff / samplerate);
357     if (r > 0.45) {
358         if (wet == 0) {
359             if (resonancedB < 0.00001)
360                 wet = 0.0f;
361             else
362                 wet = 1.0f;
363         }
364         r = 0.45;
365     } else
366         wet = 1.0f;
367
368     double c = 1.0 / (Math.tan(Math.PI * r));
369     double csq = c * c;
370     double resonance = Math.pow(10.0, -(resonancedB / 20));

```

```

371     double q = Math.sqrt(2.0f) * resonance;
372     double a0 = 1.0 / (1.0 + (q * c) + (csq));
373     double a1 = 2.0 * a0;
374     double a2 = a0;
375     double b1 = (2.0 * a0) * (1.0 - csq);
376     double b2 = a0 * (1.0 - (q * c) + csq);
377
378     this.a0 = (float)a0;
379     this.a1 = (float)a1;
380     this.a2 = (float)a2;
381     this.b1 = (float)b1;
382     this.b2 = (float)b2;
383
384 }
385
386 if (filtertype == FILTERTYPE_HP12 || filtertype == FILTERTYPE_HP24) {
387     double r = (cutoff / samplerate);
388     if (r > 0.45)
389         r = 0.45;
390     if (r < 0.0001)
391         r = 0.0001;
392     wet = 1.0f;
393     double c = (Math.tan(Math.PI * (r)));
394     double csq = c * c;
395     double resonance = Math.pow(10.0, -(resonancedB / 20));
396     double q = Math.sqrt(2.0f) * resonance;
397     double a0 = 1.0 / (1.0 + (q * c) + (csq));
398     double a1 = -2.0 * a0;
399     double a2 = a0;
400     double b1 = (2.0 * a0) * (csq - 1.0);
401     double b2 = a0 * (1.0 - (q * c) + csq);
402
403     this.a0 = (float)a0;
404     this.a1 = (float)a1;
405     this.a2 = (float)a2;
406     this.b1 = (float)b1;
407     this.b2 = (float)b2;
408
409 }
410
411 }
412
413 public void filter2(SoftAudioBuffer sbuffer) {
414
415     float[] buffer = sbuffer.array();
416
417     if (dirty) {
418         filter2calc();
419         dirty = false;
420     }
421     if (!last_set) {
422         last_a0 = a0;
423         last_a1 = a1;
424         last_a2 = a2;
425         last_b1 = b1;
426         last_b2 = b2;
427         last_q = q;
428         last_gain = gain;
429         last_wet = wet;
430         last_set = true;
431     }
432

```

```

433     if (wet > 0 || last_wet > 0) {
434
435         int len = buffer.length;
436         float a0 = this.last_a0;
437         float a1 = this.last_a1;
438         float a2 = this.last_a2;
439         float b1 = this.last_b1;
440         float b2 = this.last_b2;
441         float gain = this.last_gain;
442         float wet = this.last_wet;
443         float a0_delta = (this.a0 - this.last_a0) / len;
444         float a1_delta = (this.a1 - this.last_a1) / len;
445         float a2_delta = (this.a2 - this.last_a2) / len;
446         float b1_delta = (this.b1 - this.last_b1) / len;
447         float b2_delta = (this.b2 - this.last_b2) / len;
448         float gain_delta = (this.gain - this.last_gain) / len;
449         float wet_delta = (this.wet - this.last_wet) / len;
450         float x1 = this.x1;
451         float x2 = this.x2;
452         float y1 = this.y1;
453         float y2 = this.y2;
454
455         if (wet_delta != 0) {
456             for (int i = 0; i < len; i++) {
457                 a0 += a0_delta;
458                 a1 += a1_delta;
459                 a2 += a2_delta;
460                 b1 += b1_delta;
461                 b2 += b2_delta;
462                 gain += gain_delta;
463                 wet += wet_delta;
464                 float x = buffer[i];
465                 float y = (a0*x + a1*x1 + a2*x2 - b1*y1 - b2*y2);
466                 buffer[i] = (y * gain) * wet + (x) * (1 - wet);
467                 x2 = x1;
468                 x1 = x;
469                 y2 = y1;
470                 y1 = y;
471             }
472         } else if (a0_delta == 0 && a1_delta == 0 && a2_delta == 0
473                 && b1_delta == 0 && b2_delta == 0) {
474             for (int i = 0; i < len; i++) {
475                 float x = buffer[i];
476                 float y = (a0*x + a1*x1 + a2*x2 - b1*y1 - b2*y2);
477                 buffer[i] = y * gain;
478                 x2 = x1;
479                 x1 = x;
480                 y2 = y1;
481                 y1 = y;
482             }
483         } else {
484             for (int i = 0; i < len; i++) {
485                 a0 += a0_delta;
486                 a1 += a1_delta;
487                 a2 += a2_delta;
488                 b1 += b1_delta;
489                 b2 += b2_delta;
490                 gain += gain_delta;
491                 float x = buffer[i];
492                 float y = (a0*x + a1*x1 + a2*x2 - b1*y1 - b2*y2);
493                 buffer[i] = y * gain;
494                 x2 = x1;

```

```

495         x1 = x;
496         y2 = y1;
497         y1 = y;
498     }
499 }
500
501     if (Math.abs(x1) < 1.0E-8)
502         x1 = 0;
503     if (Math.abs(x2) < 1.0E-8)
504         x2 = 0;
505     if (Math.abs(y1) < 1.0E-8)
506         y1 = 0;
507     if (Math.abs(y2) < 1.0E-8)
508         y2 = 0;
509     this.x1 = x1;
510     this.x2 = x2;
511     this.y1 = y1;
512     this.y2 = y2;
513 }
514
515     this.last_a0 = this.a0;
516     this.last_a1 = this.a1;
517     this.last_a2 = this.a2;
518     this.last_b1 = this.b1;
519     this.last_b2 = this.b2;
520     this.last_q = this.q;
521     this.last_gain = this.gain;
522     this.last_wet = this.wet;
523
524 }
525
526 public void filter1calc() {
527     if (cutoff < 120)
528         cutoff = 120;
529     double c = (7.0 / 6.0) * Math.PI * 2 * cutoff / samplerate;
530     if (c > 1)
531         c = 1;
532     a0 = (float)(Math.sqrt(1 - Math.cos(c)) * Math.sqrt(0.5 * Math.PI));
533     if (resonancedB < 0)
534         resonancedB = 0;
535     if (resonancedB > 20)
536         resonancedB = 20;
537     q = (float)(Math.sqrt(0.5) * Math.pow(10.0, -(resonancedB / 20)));
538     gain = (float)Math.pow(10, -((resonancedB) / 40.0));
539     if (wet == 0.0f)
540         if (resonancedB > 0.00001 || c < 0.9999999)
541             wet = 1.0f;
542 }
543
544 public void filter1(SoftAudioBuffer sbuffer) {
545
546     if (dirty) {
547         filter1calc();
548         dirty = false;
549     }
550     if (!last_set) {
551         last_a0 = a0;
552         last_q = q;
553         last_gain = gain;
554         last_wet = wet;
555         last_set = true;
556     }

```

```

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

```
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 javax.sound.midi.Instrument;
28 import javax.sound.midi.MidiChannel;
29
30 /**
31 * Software synthesizer internal instrument.
32 *
33 * @author Karl Helgason
34 */
35 public class SoftInstrument extends Instrument {
36
37     private SoftPerformer[] performers;
38     private ModelPerformer[] modelperformers;
39     private Object data;
40     private ModelInstrument ins;
41
42     public SoftInstrument(ModelInstrument ins) {
43         super(ins.getSoundbank(), ins.getPatch(), ins.getName(),
44             ins.getDataClass());
45         data = ins.getData();
46         this.ins = ins;
47         initPerformers(((ModelInstrument)ins).getPerformers());
48     }
49
50     public SoftInstrument(ModelInstrument ins,
51         ModelPerformer[] overrideperformers) {
52         super(ins.getSoundbank(), ins.getPatch(), ins.getName(),
53             ins.getDataClass());
54         data = ins.getData();
55         this.ins = ins;
56         initPerformers(overrideperformers);
57     }
58
59     private void initPerformers(ModelPerformer[] modelperformers) {
60         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 }

```



```

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.io.EOFException;
28 import java.io.IOException;
29 import java.io.InputStream;
30
31 import javax.sound.sampled.AudioFormat;
32 import javax.sound.sampled.AudioInputStream;
33
34 /**
35  * A jitter corrector to be used with SoftAudioPusher.
36  *
37  * @author Karl Helgason
38  */
39 public class SoftJitterCorrector extends AudioInputStream {
40
41     private static class JitterStream extends InputStream {
42
43         static int MAX_BUFFER_SIZE = 1048576;
44         boolean active = true;
45         Thread thread;
46         AudioInputStream stream;
47         // Cyclic buffer
48         int writepos = 0;
49         int readpos = 0;
50         byte[][] buffers;
51         Object buffers_mutex = new Object();
52
53         // Adapative Drift Statistics
54         int w_count = 1000;
55         int w_min_tol = 2;
56         int w_max_tol = 10;
57         int w = 0;
58         int w_min = -1;
59         // Current read buffer
60         int bbuffer_pos = 0;

```

```

61     int bbuffer_max = 0;
62     byte[] bbuffer = null;
63
64     public byte[] nextReadBuffer() {
65         synchronized (buffers_mutex) {
66             if (writepos > readpos) {
67                 int w_m = writepos - readpos;
68                 if (w_m < w_min)
69                     w_min = w_m;
70
71                 int buffpos = readpos;
72                 readpos++;
73                 return buffers[buffpos % buffers.length];
74             }
75             w_min = -1;
76             w = w_count - 1;
77         }
78         while (true) {
79             try {
80                 Thread.sleep(1);
81             } catch (InterruptedException e) {
82                 //e.printStackTrace();
83                 return null;
84             }
85             synchronized (buffers_mutex) {
86                 if (writepos > readpos) {
87                     w = 0;
88                     w_min = -1;
89                     w = w_count - 1;
90                     int buffpos = readpos;
91                     readpos++;
92                     return buffers[buffpos % buffers.length];
93                 }
94             }
95         }
96     }
97
98     public byte[] nextWriteBuffer() {
99         synchronized (buffers_mutex) {
100             return buffers[writepos % buffers.length];
101         }
102     }
103
104     public void commit() {
105         synchronized (buffers_mutex) {
106             writepos++;
107             if ((writepos - readpos) > buffers.length) {
108                 int newsize = (writepos - readpos) + 10;
109                 newsize = Math.max(buffers.length * 2, newsize);
110                 buffers = new byte[newsize][buffers[0].length];
111             }
112         }
113     }
114
115     public JitterStream(AudioInputStream s, int buffersize,
116                         int smallbuffersize) {
117         this.w_count = 10 * (buffersize / smallbuffersize);
118         if (w_count < 100)
119             w_count = 100;
120         this.buffers
121             = new byte[(buffersize/smallbuffersize)+10][smallbuffersize];
122         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;

```

```

185         while (n != buff.length) {
186             int s = stream.read(buff, n, buff.length
187                 - n);
188             if (s < 0)
189                 throw new EOFException();
190             if (s == 0)
191                 Thread.yield();
192             n += s;
193         }
194     } catch (IOException e1) {
195         //e1.printStackTrace();
196     }
197     commit();
198 }
199
200     if (correction > 0) {
201         correction--;
202         next = System.nanoTime() + nanos;
203         continue;
204     }
205     long wait = next - System.nanoTime();
206     if (wait > 0) {
207         try {
208             Thread.sleep(wait / 1000000L);
209         } catch (InterruptedException e) {
210             //e.printStackTrace();
211         }
212     }
213     next += nanos;
214 }
215 }
216 };
217
218     thread = new Thread(runnable);
219     thread.setDaemon(true);
220     thread.setPriority(Thread.MAX_PRIORITY);
221     thread.start();
222 }
223
224 public void close() throws IOException {
225     synchronized (this) {
226         active = false;
227     }
228     try {
229         thread.join();
230     } catch (InterruptedException e) {
231         //e.printStackTrace();
232     }
233     stream.close();
234 }
235
236 public int read() throws IOException {
237     byte[] b = new byte[1];
238     if (read(b) == -1)
239         return -1;
240     return b[0] & 0xFF;
241 }
242
243 public void fillBuffer() {
244     bbuffer = nextReadBuffer();
245     bbuffer_pos = 0;
246 }

```

```

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 }

```

```

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 /**
28 * Lanczos interpolation resampler.
29 *
30 * @author Karl Helgason
31 */
32 public class SoftLanczosResampler extends SoftAbstractResampler {
33
34     float[][] sinc_table;
35     int sinc_table_fsize = 2000;
36     int sinc_table_size = 5;
37     int sinc_table_center = sinc_table_size / 2;
38
39     public SoftLanczosResampler() {
40         super();
41         sinc_table = new float[sinc_table_fsize][];
42         for (int i = 0; i < sinc_table_fsize; i++) {
43             sinc_table[i] = sincTable(sinc_table_size, -i
44                                     / ((float) sinc_table_fsize));
45         }
46     }
47
48     // Normalized sinc function
49     public static double sinc(double x) {
50         return (x == 0.0) ? 1.0 : Math.sin(Math.PI * x) / (Math.PI * x);
51     }
52
53     // Generate sinc table
54     public static float[] sincTable(int size, float offset) {
55         int center = size / 2;
56         float[] w = new float[size];
57         for (int k = 0; k < size; k++) {
58             float x = (-center + k + offset);
59             if (x < -2 || x > 2)
60                 w[k] = 0;

```

```

61         else if (x == 0)
62             w[k] = 1;
63         else {
64             w[k] = (float)(2.0 * Math.sin(Math.PI * x)
65                         * Math.sin(Math.PI * x / 2.0)
66                         / ((Math.PI * x) * (Math.PI * x)));
67         }
68     }
69     return w;
70 }
71
72 public int getPadding() // must be at least half of sinc_table_size
73 {
74     return sinc_table_size / 2 + 2;
75 }
76
77 public void interpolate(float[] in, float[] in_offset, float in_end,
78                       float[] startpitch, float pitchstep, float[] out, int[] out_offset,
79                       int out_end) {
80     float pitch = startpitch[0];
81     float ix = in_offset[0];
82     int ox = out_offset[0];
83     float ix_end = in_end;
84     int ox_end = out_end;
85
86     if (pitchstep == 0) {
87         while (ix < ix_end && ox < ox_end) {
88             int iix = (int) ix;
89             float[] sinc_table
90                 = this.sinc_table[(int) ((ix - iix) * sinc_table_fsize)];
91             int xx = iix - sinc_table_center;
92             float y = 0;
93             for (int i = 0; i < sinc_table_size; i++, xx++)
94                 y += in[xx] * sinc_table[i];
95             out[ox++] = y;
96             ix += pitch;
97         }
98     } else {
99         while (ix < ix_end && ox < ox_end) {
100             int iix = (int) ix;
101             float[] sinc_table
102                 = this.sinc_table[(int) ((ix - iix) * sinc_table_fsize)];
103             int xx = iix - sinc_table_center;
104             float y = 0;
105             for (int i = 0; i < sinc_table_size; i++, xx++)
106                 y += in[xx] * sinc_table[i];
107             out[ox++] = y;
108
109             ix += pitch;
110             pitch += pitchstep;
111         }
112     }
113     in_offset[0] = ix;
114     out_offset[0] = ox;
115     startpitch[0] = pitch;
116 }
117 }
118 }

```

91 com/sun/media/sound/SoftLimiter.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 /**
28 * A simple look-ahead volume limiter with very fast attack and fast release.
29 * This filter is used for preventing clipping.
30 *
31 * @author Karl Helgason
32 */
33 public class SoftLimiter implements SoftAudioProcessor {
34
35     float lastmax = 0;
36     float gain = 1;
37     float[] temp_bufferL;
38     float[] temp_bufferR;
39     boolean mix = false;
40     SoftAudioBuffer bufferL;
41     SoftAudioBuffer bufferR;
42     SoftAudioBuffer bufferLout;
43     SoftAudioBuffer bufferRout;
44     float controlrate;
45
46     public void init(float samplerate, float controlrate) {
47         this.controlrate = controlrate;
48     }
49
50     public void setInput(int pin, SoftAudioBuffer input) {
51         if (pin == 0)
52             bufferL = input;
53         if (pin == 1)
54             bufferR = input;
55     }
56
57     public void setOutput(int pin, SoftAudioBuffer output) {
58         if (pin == 0)
59             bufferLout = output;
60         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     float lmax = lastmax;
124     lastmax = max;
125     if (lmax > max)
126         max = lmax;
127
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     } else {
162         if (bufferR == null) {
163             for (int i = 0; i < len; i++) {
164                 gain += gaindelta;
165                 float bL = bufferL[i];
166                 float tL = temp_bufferL[i];
167                 temp_bufferL[i] = bL;
168                 bufferLout[i] = tL * gain;
169             }
170         } else {
171             for (int i = 0; i < len; i++) {
172                 gain += gaindelta;
173                 float bL = bufferL[i];
174                 float bR = bufferR[i];
175                 float tL = temp_bufferL[i];
176                 float tR = temp_bufferR[i];
177                 temp_bufferL[i] = bL;
178                 temp_bufferR[i] = bR;
179                 bufferLout[i] = tL * gain;
180                 bufferRout[i] = tR * gain;
181             }
182         }
183     }
184

```

```
185     }
186     gain = newgain;
187 }
188
189 public void processControlLogic() {
190 }
191 }
```

92 com/sun/media/sound/SoftLinearResampler.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 /**
28 * A resampler that uses first-order (linear) interpolation.
29 *
30 * @author Karl Helgason
31 */
32 public class SoftLinearResampler extends SoftAbstractResampler {
33
34     public int getPadding() {
35         return 2;
36     }
37
38     public void interpolate(float[] in, float[] in_offset, float in_end,
39         float[] startpitch, float pitchstep, float[] out, int[] out_offset,
40         int out_end) {
41
42         float pitch = startpitch[0];
43         float ix = in_offset[0];
44         int ox = out_offset[0];
45         float ix_end = in_end;
46         int ox_end = out_end;
47         if (pitchstep == 0f) {
48             while (ix < ix_end && ox < ox_end) {
49                 int iix = (int) ix;
50                 float fix = ix - iix;
51                 float i = in[iix];
52                 out[ox++] = i + (in[iix + 1] - i) * fix;
53                 ix += pitch;
54             }
55         } else {
56             while (ix < ix_end && ox < ox_end) {
57                 int iix = (int) ix;
58                 float fix = ix - iix;
59                 float i = in[iix];
60                 out[ox++] = i + (in[iix + 1] - i) * fix;
```

```
61         ix += pitch;
62         pitch += pitchstep;
63     }
64 }
65 in_offset[0] = ix;
66 out_offset[0] = ox;
67 startpitch[0] = pitch;
68
69 }
70 }
```

93 com/sun/media/sound/SoftLinearResampler2.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 /**
28 * A resampler that uses first-order (linear) interpolation.
29 *
30 * This one doesn't perform float to int casting inside the processing loop.
31 *
32 * @author Karl Helgason
33 */
34 public class SoftLinearResampler2 extends SoftAbstractResampler {
35
36     public int getPadding() {
37         return 2;
38     }
39
40     public void interpolate(float[] in, float[] in_offset, float in_end,
41                             float[] startpitch, float pitchstep, float[] out, int[] out_offset,
42                             int out_end) {
43
44         float pitch = startpitch[0];
45         float ix = in_offset[0];
46         int ox = out_offset[0];
47         float ix_end = in_end;
48         int ox_end = out_end;
49
50         // Check if we have do anything
51         if (!(ix < ix_end && ox < ox_end))
52             return;
53
54         // 15 bit shift was choosed because
55         // it resulted in no drift between p_ix and ix.
56         int p_ix = (int) (ix * (1 << 15));
57         int p_ix_end = (int) (ix_end * (1 << 15));
58         int p_pitch = (int) (pitch * (1 << 15));
59         // Pitch needs to recalculated
60         // to ensure no drift between p_ix and ix.
```

```

61     pitch = p_pitch * (1f / (1 << 15));
62
63     if (pitchstep == 0f) {
64
65         // To reduce
66         //     while (p_ix < p_ix_end && ox < ox_end)
67         // into
68         //     while (ox < ox_end)
69         // We need to calculate new ox_end value.
70         int p_ix_len = p_ix_end - p_ix;
71         int p_mod = p_ix_len % p_pitch;
72         if (p_mod != 0)
73             p_ix_len += p_pitch - p_mod;
74         int ox_end2 = ox + p_ix_len / p_pitch;
75         if (ox_end2 < ox_end)
76             ox_end = ox_end2;
77
78         while (ox < ox_end) {
79             int iix = p_ix >> 15;
80             float fix = ix - iix;
81             float i = in[iix];
82             out[ox++] = i + (in[iix + 1] - i) * fix;
83             p_ix += p_pitch;
84             ix += pitch;
85         }
86
87     } else {
88
89         int p_pitchstep = (int) (pitchstep * (1 << 15));
90         pitchstep = p_pitchstep * (1f / (1 << 15));
91
92         while (p_ix < p_ix_end && ox < ox_end) {
93             int iix = p_ix >> 15;
94             float fix = ix - iix;
95             float i = in[iix];
96             out[ox++] = i + (in[iix + 1] - i) * fix;
97             ix += pitch;
98             p_ix += p_pitch;
99             pitch += pitchstep;
100            p_pitch += p_pitchstep;
101        }
102    }
103    in_offset[0] = ix;
104    out_offset[0] = ox;
105    startpitch[0] = pitch;
106
107 }
108 }

```

94 com/sun/media/sound/SoftLowFrequencyOscillator.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 /**
28 * LFO control signal generator.
29 *
30 * @author Karl Helgason
31 */
32 public class SoftLowFrequencyOscillator implements SoftProcess {
33
34     private int max_count = 10;
35     private int used_count = 0;
36     private double[][] out = new double[max_count][1];
37     private double[][] delay = new double[max_count][1];
38     private double[][] delay2 = new double[max_count][1];
39     private double[][] freq = new double[max_count][1];
40     private int[] delay_counter = new int[max_count];
41     private double[] sin_phase = new double[max_count];
42     private double[] sin_stepfreq = new double[max_count];
43     private double[] sin_step = new double[max_count];
44     private double control_time = 0;
45     private double sin_factor = 0;
46     private static double PI2 = 2.0 * Math.PI;
47
48     public SoftLowFrequencyOscillator() {
49         // If sin_step is 0 then sin_stepfreq must be -INF
50         for (int i = 0; i < sin_stepfreq.length; i++) {
51             sin_stepfreq[i] = Double.NEGATIVE_INFINITY;
52         }
53     }
54
55     public void reset() {
56         for (int i = 0; i < used_count; i++) {
57             out[i][0] = 0;
58             delay[i][0] = 0;
59             delay2[i][0] = 0;
60             freq[i][0] = 0;
```



```

61         delay_counter[i] = 0;
62         sin_phase[i] = 0;
63         // If sin_step is 0 then sin_stepfreq must be -INF
64         sin_stepfreq[i] = Double.NEGATIVE_INFINITY;
65         sin_step[i] = 0;
66     }
67     used_count = 0;
68 }
69
70 public void init(SoftSynthesizer synth) {
71     control_time = 1.0 / synth.getControlRate();
72     sin_factor = control_time * 2 * Math.PI;
73     for (int i = 0; i < used_count; i++) {
74         delay_counter[i] = (int)(Math.pow(2,
75             this.delay[i][0] / 1200.0) / control_time);
76         delay_counter[i] += (int)(delay2[i][0] / (control_time * 1000));
77     }
78     processControlLogic();
79 }
80
81 public void processControlLogic() {
82     for (int i = 0; i < used_count; i++) {
83         if (delay_counter[i] > 0) {
84             delay_counter[i]--;
85             out[i][0] = 0.5;
86         } else {
87             double f = freq[i][0];
88
89             if (sin_stepfreq[i] != f) {
90                 sin_stepfreq[i] = f;
91                 double fr = 440.0 * Math.exp(
92                     (f - 6900.0) * (Math.log(2) / 1200.0));
93                 sin_step[i] = fr * sin_factor;
94             }
95             /*
96             double fr = 440.0 * Math.pow(2.0,
97                 (freq[i][0] - 6900.0) / 1200.0);
98             sin_phase[i] += fr * sin_factor;
99             */
100            /*
101            sin_phase[i] += sin_step[i];
102            while (sin_phase[i] > PI2)
103                sin_phase[i] -= PI2;
104            out[i][0] = 0.5 + Math.sin(sin_phase[i]) * 0.5;
105            */
106            double p = sin_phase[i];
107            p += sin_step[i];
108            while (p > PI2)
109                p -= PI2;
110            out[i][0] = 0.5 + Math.sin(p) * 0.5;
111            sin_phase[i] = p;
112        }
113    }
114 }
115 }
116
117 public double[] get(int instance, String name) {
118     if (instance >= used_count)
119         used_count = instance + 1;
120     if (name == null)
121         return out[instance];
122     if (name.equals("delay"))

```

```
123         return delay[instance];
124     if (name.equals("delay2"))
125         return delay2[instance];
126     if (name.equals("freq"))
127         return freq[instance];
128     return null;
129 }
130 }
```

95 com/sun/media/sound/SoftMainMixer.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.io.IOException;
28 import java.io.InputStream;
29 import java.util.HashSet;
30 import java.util.Iterator;
31 import java.util.Set;
32 import java.util.TreeMap;
33 import java.util.Map.Entry;
34
35 import javax.sound.midi.MidiMessage;
36 import javax.sound.midi.Patch;
37 import javax.sound.midi.ShortMessage;
38 import javax.sound.sampled.AudioInputStream;
39 import javax.sound.sampled.AudioSystem;
40
41 /**
42 * Software synthesizer main audio mixer.
43 *
44 * @author Karl Helgason
45 */
46 public class SoftMainMixer {
47
48     // A private class thats contains a ModelChannelMixer and it's private buffers.
49     // This becomes necessary when we want to have separate delay buffers for each channel mixer.
50     private class SoftChannelMixerContainer
51     {
52         ModelChannelMixer mixer;
53         SoftAudioBuffer[] buffers;
54     }
55
56     public final static int CHANNEL_LEFT = 0;
57     public final static int CHANNEL_RIGHT = 1;
58     public final static int CHANNEL_MONO = 2;
59     public final static int CHANNEL_DELAY_LEFT = 3;
60     public final static int CHANNEL_DELAY_RIGHT = 4;
```

```

61 public final static int CHANNEL_DELAY_MONO = 5;
62 public final static int CHANNEL_EFFECT1 = 6;
63 public final static int CHANNEL_EFFECT2 = 7;
64 public final static int CHANNEL_DELAY_EFFECT1 = 8;
65 public final static int CHANNEL_DELAY_EFFECT2 = 9;
66 public final static int CHANNEL_LEFT_DRY = 10;
67 public final static int CHANNEL_RIGHT_DRY = 11;
68 public final static int CHANNEL_SCRATCH1 = 12;
69 public final static int CHANNEL_SCRATCH2 = 13;
70 protected boolean active_sensing_on = false;
71 private long msec_last_activity = -1;
72 private boolean pusher_silent = false;
73 private int pusher_silent_count = 0;
74 private long sample_pos = 0;
75 protected boolean readfully = true;
76 private Object control_mutex;
77 private SoftSynthesizer synth;
78 private float samplerate = 44100;
79 private int nrofchannels = 2;
80 private SoftVoice[] voicestatus = null;
81 private SoftAudioBuffer[] buffers;
82 private SoftReverb reverb;
83 private SoftAudioProcessor chorus;
84 private SoftAudioProcessor agc;
85 private long msec_buffer_len = 0;
86 private int buffer_len = 0;
87 protected TreeMap<Long, Object> midimessages = new TreeMap<Long, Object>();
88 private int delay_midievent = 0;
89 private int max_delay_midievent = 0;
90 double last_volume_left = 1.0;
91 double last_volume_right = 1.0;
92 private double[] co_master_balance = new double[1];
93 private double[] co_master_volume = new double[1];
94 private double[] co_master_coarse_tuning = new double[1];
95 private double[] co_master_fine_tuning = new double[1];
96 private AudioInputStream ais;
97 private Set<SoftChannelMixerContainer> registeredMixers = null;
98 private Set<ModelChannelMixer> stoppedMixers = null;
99 private SoftChannelMixerContainer[] cur_registeredMixers = null;
100 protected SoftControl co_master = new SoftControl() {
101
102     double[] balance = co_master_balance;
103     double[] volume = co_master_volume;
104     double[] coarse_tuning = co_master_coarse_tuning;
105     double[] fine_tuning = co_master_fine_tuning;
106
107     public double[] get(int instance, String name) {
108         if (name == null)
109             return null;
110         if (name.equals("balance"))
111             return balance;
112         if (name.equals("volume"))
113             return volume;
114         if (name.equals("coarse_tuning"))
115             return coarse_tuning;
116         if (name.equals("fine_tuning"))
117             return fine_tuning;
118         return null;
119     }
120 };
121
122 private void processSystemExclusiveMessage(byte[] data) {

```

```

123 synchronized (synth.control_mutex) {
124     activity();
125
126     // Universal Non-Real-Time SysEx
127     if ((data[1] & 0xFF) == 0x7E) {
128         int deviceID = data[2] & 0xFF;
129         if (deviceID == 0x7F || deviceID == synth.getDeviceID()) {
130             int subid1 = data[3] & 0xFF;
131             int subid2;
132             switch (subid1) {
133                 case 0x08: // MIDI Tuning Standard
134                     subid2 = data[4] & 0xFF;
135                     switch (subid2) {
136                         case 0x01: // BULK TUNING DUMP
137                         {
138                             // http://www.midi.org/about-midi/tuning.shtml
139                             SoftTuning tuning = synth.getTuning(new Patch(0,
140                                 data[5] & 0xFF));
141                             tuning.load(data);
142                             break;
143                         }
144                         case 0x04: // KEY-BASED TUNING DUMP
145                         case 0x05: // SCALE/OCTAVE TUNING DUMP, 1 byte format
146                         case 0x06: // SCALE/OCTAVE TUNING DUMP, 2 byte format
147                         case 0x07: // SINGLE NOTE TUNING CHANGE (NON REAL-TIME)
148                             // (BANK)
149                         {
150                             // http://www.midi.org/about-midi/tuning_extens.shtml
151                             SoftTuning tuning = synth.getTuning(new Patch(
152                                 data[5] & 0xFF, data[6] & 0xFF));
153                             tuning.load(data);
154                             break;
155                         }
156                         case 0x08: // scale/octave tuning 1-byte form (Non
157                             // Real-Time)
158                         case 0x09: // scale/octave tuning 2-byte form (Non
159                             // Real-Time)
160                         {
161                             // http://www.midi.org/about-midi/tuning-scale.shtml
162                             SoftTuning tuning = new SoftTuning(data);
163                             int channelmask = (data[5] & 0xFF) * 16384
164                                 + (data[6] & 0xFF) * 128 + (data[7] & 0xFF);
165                             SoftChannel[] channels = synth.channels;
166                             for (int i = 0; i < channels.length; i++)
167                                 if ((channelmask & (1 << i)) != 0)
168                                     channels[i].tuning = tuning;
169                             break;
170                         }
171                         default:
172                             break;
173                     }
174                     break;
175                 case 0x09: // General Midi Message
176                     subid2 = data[4] & 0xFF;
177                     switch (subid2) {
178                         case 0x01: // General Midi 1 On
179                             synth.setGeneralMidiMode(1);
180                             reset();
181                             break;
182                         case 0x02: // General Midi Off
183                             synth.setGeneralMidiMode(0);
184                             reset();

```

```

185         break;
186     case 0x03: // General Midi Level 2 On
187         synth.setGeneralMidiMode(2);
188         reset();
189         break;
190     default:
191         break;
192     }
193     break;
194 case 0x0A: // DLS Message
195     subid2 = data[4] & 0xFF;
196     switch (subid2) {
197     case 0x01: // DLS On
198         if (synth.getGeneralMidiMode() == 0)
199             synth.setGeneralMidiMode(1);
200         synth.voice_allocation_mode = 1;
201         reset();
202         break;
203     case 0x02: // DLS Off
204         synth.setGeneralMidiMode(0);
205         synth.voice_allocation_mode = 0;
206         reset();
207         break;
208     case 0x03: // DLS Static Voice Allocation Off
209         synth.voice_allocation_mode = 0;
210         break;
211     case 0x04: // DLS Static Voice Allocation On
212         synth.voice_allocation_mode = 1;
213         break;
214     default:
215         break;
216     }
217     break;
218
219     default:
220         break;
221 }
222 }
223 }
224
225 // Universal Real-Time SysEx
226 if ((data[1] & 0xFF) == 0x7F) {
227     int deviceID = data[2] & 0xFF;
228     if (deviceID == 0x7F || deviceID == synth.getDeviceID()) {
229         int subid1 = data[3] & 0xFF;
230         int subid2;
231         switch (subid1) {
232         case 0x04: // Device Control
233
234             subid2 = data[4] & 0xFF;
235             switch (subid2) {
236             case 0x01: // Master Volume
237             case 0x02: // Master Balance
238             case 0x03: // Master fine tuning
239             case 0x04: // Master coarse tuning
240                 int val = (data[5] & 0x7F)
241                     + ((data[6] & 0x7F) * 128);
242                 if (subid2 == 0x01)
243                     setVolume(val);
244                 else if (subid2 == 0x02)
245                     setBalance(val);
246                 else if (subid2 == 0x03)

```

```

247         setFineTuning(val);
248     else if (subid2 == 0x04)
249         setCoarseTuning(val);
250     break;
251 case 0x05: // Global Parameter Control
252     int ix = 5;
253     int slotPathLen = (data[ix++] & 0xFF);
254     int paramWidth = (data[ix++] & 0xFF);
255     int valueWidth = (data[ix++] & 0xFF);
256     int[] slotPath = new int[slotPathLen];
257     for (int i = 0; i < slotPathLen; i++) {
258         int msb = (data[ix++] & 0xFF);
259         int lsb = (data[ix++] & 0xFF);
260         slotPath[i] = msb * 128 + lsb;
261     }
262     int paramCount = (data.length - 1 - ix)
263         / (paramWidth + valueWidth);
264     long[] params = new long[paramCount];
265     long[] values = new long[paramCount];
266     for (int i = 0; i < paramCount; i++) {
267         values[i] = 0;
268         for (int j = 0; j < paramWidth; j++)
269             params[i] = params[i] * 128
270                 + (data[ix++] & 0xFF);
271         for (int j = 0; j < valueWidth; j++)
272             values[i] = values[i] * 128
273                 + (data[ix++] & 0xFF);
274     }
275     globalParameterControlChange(slotPath, params, values);
276     break;
277 default:
278     break;
279 }
280 break;
281
282
283 case 0x08: // MIDI Tuning Standard
284     subid2 = data[4] & 0xFF;
285     switch (subid2) {
286     case 0x02: // SINGLE NOTE TUNING CHANGE (REAL-TIME)
287     {
288         // http://www.midi.org/about-midi/tuning.shtml
289         SoftTuning tuning = synth.getTuning(new Patch(0,
290             data[5] & 0xFF));
291         tuning.load(data);
292         SoftVoice[] voices = synth.getVoices();
293         for (int i = 0; i < voices.length; i++)
294             if (voices[i].active)
295                 if (voices[i].tuning == tuning)
296                     voices[i].updateTuning(tuning);
297         break;
298     }
299     case 0x07: // SINGLE NOTE TUNING CHANGE (REAL-TIME)
300                 // (BANK)
301     {
302         // http://www.midi.org/about-midi/tuning_extens.shtml
303         SoftTuning tuning = synth.getTuning(new Patch(
304             data[5] & 0xFF, data[6] & 0xFF));
305         tuning.load(data);
306         SoftVoice[] voices = synth.getVoices();
307         for (int i = 0; i < voices.length; i++)
308             if (voices[i].active)

```

```

309         if (voices[i].tuning == tuning)
310             voices[i].updateTuning(tuning);
311     break;
312 }
313 case 0x08: // scale/octave tuning 1-byte form
314             //(Real-Time)
315 case 0x09: // scale/octave tuning 2-byte form
316             //(Real-Time)
317 {
318     // http://www.midi.org/about-midi/tuning-scale.shtml
319     SoftTuning tuning = new SoftTuning(data);
320     int channelmask = (data[5] & 0xFF) * 16384
321                     + (data[6] & 0xFF) * 128 + (data[7] & 0xFF);
322     SoftChannel[] channels = synth.channels;
323     for (int i = 0; i < channels.length; i++)
324         if ((channelmask & (1 << i)) != 0)
325             channels[i].tuning = tuning;
326     SoftVoice[] voices = synth.getVoices();
327     for (int i = 0; i < voices.length; i++)
328         if (voices[i].active)
329             if ((channelmask & (1 << (voices[i].channel))) != 0)
330                 voices[i].updateTuning(tuning);
331     break;
332 }
333 default:
334     break;
335 }
336 break;
337 case 0x09: // Control Destination Settings
338     subid2 = data[4] & 0xFF;
339     switch (subid2) {
340     case 0x01: // Channel Pressure
341     {
342         int[] destinations = new int[(data.length - 7) / 2];
343         int[] ranges = new int[(data.length - 7) / 2];
344         int ix = 0;
345         for (int j = 6; j < data.length - 1; j += 2) {
346             destinations[ix] = data[j] & 0xFF;
347             ranges[ix] = data[j + 1] & 0xFF;
348             ix++;
349         }
350         int channel = data[5] & 0xFF;
351         SoftChannel softchannel = synth.channels[channel];
352         softchannel.mapChannelPressureToDestination(
353             destinations, ranges);
354         break;
355     }
356     case 0x02: // Poly Pressure
357     {
358         int[] destinations = new int[(data.length - 7) / 2];
359         int[] ranges = new int[(data.length - 7) / 2];
360         int ix = 0;
361         for (int j = 6; j < data.length - 1; j += 2) {
362             destinations[ix] = data[j] & 0xFF;
363             ranges[ix] = data[j + 1] & 0xFF;
364             ix++;
365         }
366         int channel = data[5] & 0xFF;
367         SoftChannel softchannel = synth.channels[channel];
368         softchannel.mapPolyPressureToDestination(
369             destinations, ranges);
370         break;

```



```

371     }
372     case 0x03: // Control Change
373     {
374         int[] destinations = new int[(data.length - 7) / 2];
375         int[] ranges = new int[(data.length - 7) / 2];
376         int ix = 0;
377         for (int j = 7; j < data.length - 1; j += 2) {
378             destinations[ix] = data[j] & 0xFF;
379             ranges[ix] = data[j + 1] & 0xFF;
380             ix++;
381         }
382         int channel = data[5] & 0xFF;
383         SoftChannel softchannel = synth.channels[channel];
384         int control = data[6] & 0xFF;
385         softchannel.mapControlToDestination(control,
386             destinations, ranges);
387         break;
388     }
389     default:
390         break;
391     }
392     break;
393
394     case 0x0A: // Key Based Instrument Control
395     {
396         subid2 = data[4] & 0xFF;
397         switch (subid2) {
398             case 0x01: // Basic Message
399             {
400                 int channel = data[5] & 0xFF;
401                 int keynumber = data[6] & 0xFF;
402                 SoftChannel softchannel = synth.channels[channel];
403                 for (int j = 7; j < data.length - 1; j += 2) {
404                     int controlnumber = data[j] & 0xFF;
405                     int controlvalue = data[j + 1] & 0xFF;
406                     softchannel.controlChangePerNote(keynumber,
407                         controlnumber, controlvalue);
408                 }
409                 break;
410             default:
411                 break;
412             }
413             break;
414         }
415         default:
416             break;
417     }
418 }
419
420 }
421
422
423 private void processMessages(long timeStamp) {
424     Iterator<Entry<Long, Object>> iter = midimessages.entrySet().iterator();
425     while (iter.hasNext()) {
426         Entry<Long, Object> entry = iter.next();
427         if (entry.getKey() >= (timeStamp + msec_buffer_len))
428             return;
429         long msec_delay = entry.getKey() - timeStamp;
430         delay_midievent = (int)(msec_delay * (samplerate / 1000000.0) + 0.5);
431         if (delay_midievent > max_delay_midievent)
432             delay_midievent = max_delay_midievent;

```

```

433         if(delay_midievent < 0)
434             delay_midievent = 0;
435         processMessage(entry.getValue());
436         iter.remove();
437     }
438     delay_midievent = 0;
439 }
440
441 protected void processAudioBuffers() {
442
443     if(synth.weakstream != null && synth.weakstream.silent_samples != 0)
444     {
445         sample_pos += synth.weakstream.silent_samples;
446         synth.weakstream.silent_samples = 0;
447     }
448
449     for (int i = 0; i < buffers.length; i++) {
450         if(i != CHANNEL_DELAY_LEFT &&
451            i != CHANNEL_DELAY_RIGHT &&
452            i != CHANNEL_DELAY_MONO &&
453            i != CHANNEL_DELAY_EFFECT1 &&
454            i != CHANNEL_DELAY_EFFECT2)
455             buffers[i].clear();
456     }
457
458     if(!buffers[CHANNEL_DELAY_LEFT].isSilent())
459     {
460         buffers[CHANNEL_LEFT].swap(buffers[CHANNEL_DELAY_LEFT]);
461     }
462     if(!buffers[CHANNEL_DELAY_RIGHT].isSilent())
463     {
464         buffers[CHANNEL_RIGHT].swap(buffers[CHANNEL_DELAY_RIGHT]);
465     }
466     if(!buffers[CHANNEL_DELAY_MONO].isSilent())
467     {
468         buffers[CHANNEL_MONO].swap(buffers[CHANNEL_DELAY_MONO]);
469     }
470     if(!buffers[CHANNEL_DELAY_EFFECT1].isSilent())
471     {
472         buffers[CHANNEL_EFFECT1].swap(buffers[CHANNEL_DELAY_EFFECT1]);
473     }
474     if(!buffers[CHANNEL_DELAY_EFFECT2].isSilent())
475     {
476         buffers[CHANNEL_EFFECT2].swap(buffers[CHANNEL_DELAY_EFFECT2]);
477     }
478
479     double volume_left;
480     double volume_right;
481
482     SoftChannelMixerContainer[] act_registeredMixers;
483
484     // perform control logic
485     synchronized (control_mutex) {
486
487         long msec_pos = (long)(sample_pos * (1000000.0 / samplerate));
488
489         processMessages(msec_pos);
490
491         if (active_sensing_on) {
492             // Active Sensing
493             // if no message occurs for max 1000 ms
494             // then do AllSoundOff on all channels

```

```

495         if ((msec_pos - msec_last_activity) > 1000000) {
496             active_sensing_on = false;
497             for (SoftChannel c : synth.channels)
498                 c.allSoundOff();
499         }
500     }
501
502     for (int i = 0; i < voicestatus.length; i++)
503         if (voicestatus[i].active)
504             voicestatus[i].processControlLogic();
505     sample_pos += buffer_len;
506
507     double volume = co_master_volume[0];
508     volume_left = volume;
509     volume_right = volume;
510
511     double balance = co_master_balance[0];
512     if (balance > 0.5)
513         volume_left *= (1 - balance) * 2;
514     else
515         volume_right *= balance * 2;
516
517     chorus.processControlLogic();
518     reverb.processControlLogic();
519     agc.processControlLogic();
520
521     if (cur_registeredMixers == null) {
522         if (registeredMixers != null) {
523             cur_registeredMixers =
524                 new SoftChannelMixerContainer[registeredMixers.size()];
525             registeredMixers.toArray(cur_registeredMixers);
526         }
527     }
528
529     act_registeredMixers = cur_registeredMixers;
530     if (act_registeredMixers != null)
531         if (act_registeredMixers.length == 0)
532             act_registeredMixers = null;
533
534 }
535
536 if (act_registeredMixers != null) {
537
538     // Make backup of left,right,mono channels
539     SoftAudioBuffer leftbak = buffers[CHANNEL_LEFT];
540     SoftAudioBuffer rightbak = buffers[CHANNEL_RIGHT];
541     SoftAudioBuffer monobak = buffers[CHANNEL_MONO];
542     SoftAudioBuffer delayleftbak = buffers[CHANNEL_DELAY_LEFT];
543     SoftAudioBuffer delayrightbak = buffers[CHANNEL_DELAY_RIGHT];
544     SoftAudioBuffer delaymonobak = buffers[CHANNEL_DELAY_MONO];
545
546     int bufferlen = buffers[CHANNEL_LEFT].getSize();
547
548     float[][] cbuffer = new float[nrofchannels][];
549     float[][] obuffer = new float[nrofchannels][];
550     obuffer[0] = leftbak.array();
551     if (nrofchannels != 1)
552         obuffer[1] = rightbak.array();
553
554     for (SoftChannelMixerContainer cmixer : act_registeredMixers) {

```

```

557 // Reroute default left,right output
558 // to channelmixer left,right input/output
559 buffers[CHANNEL_LEFT] = cmixer.buffers[CHANNEL_LEFT];
560 buffers[CHANNEL_RIGHT] = cmixer.buffers[CHANNEL_RIGHT];
561 buffers[CHANNEL_MONO] = cmixer.buffers[CHANNEL_MONO];
562 buffers[CHANNEL_DELAY_LEFT] = cmixer.buffers[CHANNEL_DELAY_LEFT];
563 buffers[CHANNEL_DELAY_RIGHT] = cmixer.buffers[CHANNEL_DELAY_RIGHT];
564 buffers[CHANNEL_DELAY_MONO] = cmixer.buffers[CHANNEL_DELAY_MONO];
565
566 buffers[CHANNEL_LEFT].clear();
567 buffers[CHANNEL_RIGHT].clear();
568 buffers[CHANNEL_MONO].clear();
569
570 if(!buffers[CHANNEL_DELAY_LEFT].isSilent())
571 {
572     buffers[CHANNEL_LEFT].swap(buffers[CHANNEL_DELAY_LEFT]);
573 }
574 if(!buffers[CHANNEL_DELAY_RIGHT].isSilent())
575 {
576     buffers[CHANNEL_RIGHT].swap(buffers[CHANNEL_DELAY_RIGHT]);
577 }
578 if(!buffers[CHANNEL_DELAY_MONO].isSilent())
579 {
580     buffers[CHANNEL_MONO].swap(buffers[CHANNEL_DELAY_MONO]);
581 }
582
583 cbuffer[0] = buffers[CHANNEL_LEFT].array();
584 if (nrofchannels != 1)
585     cbuffer[1] = buffers[CHANNEL_RIGHT].array();
586
587 boolean hasactivevoices = false;
588 for (int i = 0; i < voicestatus.length; i++)
589     if (voicestatus[i].active)
590         if (voicestatus[i].channelmixer == cmixer.mixer) {
591             voicestatus[i].processAudioLogic(buffers);
592             hasactivevoices = true;
593         }
594
595
596 if(!buffers[CHANNEL_MONO].isSilent())
597 {
598     float[] mono = buffers[CHANNEL_MONO].array();
599     float[] left = buffers[CHANNEL_LEFT].array();
600     if (nrofchannels != 1) {
601         float[] right = buffers[CHANNEL_RIGHT].array();
602         for (int i = 0; i < bufferlen; i++) {
603             float v = mono[i];
604             left[i] += v;
605             right[i] += v;
606         }
607     }
608     else
609     {
610         for (int i = 0; i < bufferlen; i++) {
611             left[i] += mono[i];
612         }
613     }
614 }
615
616 if (!cmixer.mixer.process(cbuffer, 0, bufferlen)) {
617     synchronized (control_mutex) {
618         registeredMixers.remove(cmixer);

```

```

619         cur_registeredMixers = null;
620     }
621 }
622
623 for (int i = 0; i < cbuffer.length; i++) {
624     float[] cbuff = cbuffer[i];
625     float[] obuff = obuffer[i];
626     for (int j = 0; j < bufferlen; j++)
627         obuff[j] += cbuff[j];
628 }
629
630 if (!hasactivevoices) {
631     synchronized (control_mutex) {
632         if (stoppedMixers != null) {
633             if (stoppedMixers.contains(cmixer)) {
634                 stoppedMixers.remove(cmixer);
635                 cmixer.mixer.stop();
636             }
637         }
638     }
639 }
640
641 }
642
643 buffers[CHANNEL_LEFT] = leftbak;
644 buffers[CHANNEL_RIGHT] = rightbak;
645 buffers[CHANNEL_MONO] = monobak;
646 buffers[CHANNEL_DELAY_LEFT] = delayleftbak;
647 buffers[CHANNEL_DELAY_RIGHT] = delayrightbak;
648 buffers[CHANNEL_DELAY_MONO] = delaymonobak;
649
650 }
651
652 for (int i = 0; i < voicestatus.length; i++)
653     if (voicestatus[i].active)
654         if (voicestatus[i].channelmixer == null)
655             voicestatus[i].processAudioLogic(buffers);
656
657 if(!buffers[CHANNEL_MONO].isSilent())
658 {
659     float[] mono = buffers[CHANNEL_MONO].array();
660     float[] left = buffers[CHANNEL_LEFT].array();
661     int bufferlen = buffers[CHANNEL_LEFT].getSize();
662     if (nrofchannels != 1) {
663         float[] right = buffers[CHANNEL_RIGHT].array();
664         for (int i = 0; i < bufferlen; i++) {
665             float v = mono[i];
666             left[i] += v;
667             right[i] += v;
668         }
669     }
670     else
671     {
672         for (int i = 0; i < bufferlen; i++) {
673             left[i] += mono[i];
674         }
675     }
676 }
677
678 // Run effects
679 if (synth.chorus_on)
680     chorus.processAudio();

```

```

681     if (synth.reverb_on)
682         reverb.processAudio();
683
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++;

```

```

743         if(pushersilent_count > 5)
744         {
745             pushersilent_count = 0;
746             synchronized (control_mutex) {
747                 pushersilent = true;
748                 if(synth.weakstream != null)
749                     synth.weakstream.setInputStream(null);
750             }
751         }
752     }
753 }
754 else
755     pushersilent_count = 0;
756
757 if (synth.agc_on)
758     agc.processAudio();
759
760 }
761
762 // Must only be called within control_mutex synchronization
763 public void activity()
764 {
765     long silent_samples = 0;
766     if(pushersilent)
767     {
768         pushersilent = false;
769         if(synth.weakstream != null)
770         {
771             synth.weakstream.setInputStream(ais);
772             silent_samples = synth.weakstream.silent_samples;;
773         }
774     }
775     msec_last_activity = (long)((sample_pos + silent_samples)
776                                * (1000000.0 / samplerate));
777 }
778
779 public void stopMixer(ModelChannelMixer mixer) {
780     if (stoppedMixers == null)
781         stoppedMixers = new HashSet<ModelChannelMixer>();
782     stoppedMixers.add(mixer);
783 }
784
785 public void registerMixer(ModelChannelMixer mixer) {
786     if (registeredMixers == null)
787         registeredMixers = new HashSet<SoftChannelMixerContainer>();
788     SoftChannelMixerContainer mixercontainer = new SoftChannelMixerContainer();
789     mixercontainer.buffers = new SoftAudioBuffer[6];
790     for (int i = 0; i < mixercontainer.buffers.length; i++) {
791         mixercontainer.buffers[i] =
792             new SoftAudioBuffer(buffer_len, synth.getFormat());
793     }
794     mixercontainer.mixer = mixer;
795     registeredMixers.add(mixercontainer);
796     cur_registeredMixers = null;
797 }
798
799 public SoftMainMixer(SoftSynthesizer synth) {
800     this.synth = synth;
801
802     sample_pos = 0;
803
804     co_master_balance[0] = 0.5;

```

```

805 co_master_volume[0] = 1;
806 co_master_coarse_tuning[0] = 0.5;
807 co_master_fine_tuning[0] = 0.5;
808
809 msec_buffer_len = (long) (1000000.0 / synth.getControlRate());
810 samplerate = synth.getFormat().getSampleRate();
811 nrofchannels = synth.getFormat().getChannels();
812
813 int buffersize = (int) (synth.getFormat().getSampleRate()
814                        / synth.getControlRate());
815
816 buffer_len = buffersize;
817
818 max_delay_midievent = buffersize;
819
820 control_mutex = synth.control_mutex;
821 buffers = new SoftAudioBuffer[14];
822 for (int i = 0; i < buffers.length; i++) {
823     buffers[i] = new SoftAudioBuffer(buffersize, synth.getFormat());
824 }
825 voicestatus = synth.getVoices();
826
827 reverb = new SoftReverb();
828 chorus = new SoftChorus();
829 agc = new SoftLimiter();
830
831 float samplerate = synth.getFormat().getSampleRate();
832 float controlrate = synth.getControlRate();
833 reverb.init(samplerate, controlrate);
834 chorus.init(samplerate, controlrate);
835 agc.init(samplerate, controlrate);
836
837 reverb.setLightMode(synth.reverb_light);
838
839 reverb.setMixMode(true);
840 chorus.setMixMode(true);
841 agc.setMixMode(false);
842
843 chorus.setInput(0, buffers[CHANNEL_EFFECT2]);
844 chorus.setOutput(0, buffers[CHANNEL_LEFT]);
845 if (nrofchannels != 1)
846     chorus.setOutput(1, buffers[CHANNEL_RIGHT]);
847 chorus.setOutput(2, buffers[CHANNEL_EFFECT1]);
848
849 reverb.setInput(0, buffers[CHANNEL_EFFECT1]);
850 reverb.setOutput(0, buffers[CHANNEL_LEFT]);
851 if (nrofchannels != 1)
852     reverb.setOutput(1, buffers[CHANNEL_RIGHT]);
853
854 agc.setInput(0, buffers[CHANNEL_LEFT]);
855 if (nrofchannels != 1)
856     agc.setInput(1, buffers[CHANNEL_RIGHT]);
857 agc.setOutput(0, buffers[CHANNEL_LEFT]);
858 if (nrofchannels != 1)
859     agc.setOutput(1, buffers[CHANNEL_RIGHT]);
860
861 InputStream in = new InputStream() {
862
863     private SoftAudioBuffer[] buffers = SoftMainMixer.this.buffers;
864     private int nrofchannels
865         = SoftMainMixer.this.synth.getFormat().getChannels();
866     private int buffersize = buffers[0].getSize();

```



```

867     private byte[] bbuffer = new byte[bufferSize
868         * (SoftMainMixer.this.synth.getFormat()
869             .getSampleSizeInBits() / 8)
870         * nrofchannels];
871     private int bbuffer_pos = 0;
872     private byte[] single = new byte[1];
873
874     public void fillBuffer() {
875         /*
876         boolean pusher_silent2;
877         synchronized (control_mutex) {
878             pusher_silent2 = pusher_silent;
879         }
880         if(!pusher_silent2)*//
881         processAudioBuffers();
882         for (int i = 0; i < nrofchannels; i++)
883             buffers[i].get(bbuffer, i);
884         bbuffer_pos = 0;
885     }
886
887     public int read(byte[] b, int off, int len) {
888         int bbuffer_len = bbuffer.length;
889         int offlen = off + len;
890         int orgoff = off;
891         byte[] bbuffer = this.bbuffer;
892         while (off < offlen) {
893             if (available() == 0)
894                 fillBuffer();
895             else {
896                 int bbuffer_pos = this.bbuffer_pos;
897                 while (off < offlen && bbuffer_pos < bbuffer_len)
898                     b[off++] = bbuffer[bbuffer_pos++];
899                 this.bbuffer_pos = bbuffer_pos;
900                 if (!readfully)
901                     return off - orgoff;
902             }
903         }
904         return len;
905     }
906
907     public int read() throws IOException {
908         int ret = read(single);
909         if (ret == -1)
910             return -1;
911         return single[0] & 0xFF;
912     }
913
914     public int available() {
915         return bbuffer.length - bbuffer_pos;
916     }
917
918     public void close() {
919         SoftMainMixer.this.synth.close();
920     }
921 };
922
923     ais = new AudioInputStream(in, synth.getFormat(), AudioSystem.NOT_SPECIFIED);
924
925 }
926
927 public AudioInputStream getInputStream() {
928     return ais;

```

```

929     }
930
931     public void reset() {
932
933         SoftChannel[] channels = synth.channels;
934         for (int i = 0; i < channels.length; i++) {
935             channels[i].allSoundOff();
936             channels[i].resetAllControllers(true);
937
938             if (synth.getGeneralMidiMode() == 2) {
939                 if (i == 9)
940                     channels[i].programChange(0, 0x78 * 128);
941                 else
942                     channels[i].programChange(0, 0x79 * 128);
943             } else
944                 channels[i].programChange(0, 0);
945         }
946         setVolume(0x7F * 128 + 0x7F);
947         setBalance(0x40 * 128 + 0x00);
948         setCoarseTuning(0x40 * 128 + 0x00);
949         setFineTuning(0x40 * 128 + 0x00);
950         // Reset Reverb
951         globalParameterControlChange(
952             new int[]{0x01 * 128 + 0x01}, new long[]{0}, new long[]{4});
953         // Reset Chorus
954         globalParameterControlChange(
955             new int[]{0x01 * 128 + 0x02}, new long[]{0}, new long[]{2});
956     }
957
958     public void setVolume(int value) {
959         synchronized (control_mutex) {
960             co_master_volume[0] = value / 16384.0;
961         }
962     }
963
964     public void setBalance(int value) {
965         synchronized (control_mutex) {
966             co_master_balance[0] = value / 16384.0;
967         }
968     }
969
970     public void setFineTuning(int value) {
971         synchronized (control_mutex) {
972             co_master_fine_tuning[0] = value / 16384.0;
973         }
974     }
975
976     public void setCoarseTuning(int value) {
977         synchronized (control_mutex) {
978             co_master_coarse_tuning[0] = value / 16384.0;
979         }
980     }
981
982     public int getVolume() {
983         synchronized (control_mutex) {
984             return (int) (co_master_volume[0] * 16384.0);
985         }
986     }
987
988     public int getBalance() {
989         synchronized (control_mutex) {
990             return (int) (co_master_balance[0] * 16384.0);

```

```

991     }
992 }
993
994 public int getFineTuning() {
995     synchronized (control_mutex) {
996         return (int) (co_master_fine_tuning[0] * 16384.0);
997     }
998 }
999
1000 public int getCoarseTuning() {
1001     synchronized (control_mutex) {
1002         return (int) (co_master_coarse_tuning[0] * 16384.0);
1003     }
1004 }
1005
1006 public void globalParameterControlChange(int[] slothpath, long[] params,
1007     long[] paramsvalue) {
1008     if (slothpath.length == 0)
1009         return;
1010
1011     synchronized (control_mutex) {
1012
1013         // slothpath: 01xx are reserved only for GM2
1014
1015         if (slothpath[0] == 0x01 * 128 + 0x01) {
1016             for (int i = 0; i < paramsvalue.length; i++) {
1017                 reverb.globalParameterControlChange(slothpath, params[i],
1018                     paramsvalue[i]);
1019             }
1020         }
1021         if (slothpath[0] == 0x01 * 128 + 0x02) {
1022             for (int i = 0; i < paramsvalue.length; i++) {
1023                 chorus.globalParameterControlChange(slothpath, params[i],
1024                     paramsvalue[i]);
1025             }
1026         }
1027     }
1028 }
1029 }
1030
1031
1032 public void processMessage(Object object) {
1033     if (object instanceof byte[])
1034         processMessage((byte[]) object);
1035     if (object instanceof MidiMessage)
1036         processMessage((MidiMessage) object);
1037 }
1038
1039 public void processMessage(MidiMessage message) {
1040     if (message instanceof ShortMessage) {
1041         ShortMessage sms = (ShortMessage) message;
1042         processMessage(sms.getChannel(), sms.getCommand(),
1043             sms.getData1(), sms.getData2());
1044         return;
1045     }
1046     processMessage(message.getMessage());
1047 }
1048
1049 public void processMessage(byte[] data) {
1050     int status = 0;
1051     if (data.length > 0)
1052         status = data[0] & 0xFF;

```

```

1053     if (status == 0xF0) {
1054         processSystemExclusiveMessage(data);
1055         return;
1056     }
1057
1058     int cmd = (status & 0xF0);
1059     int ch = (status & 0x0F);
1060
1061     int data1;
1062     int data2;
1063     if (data.length > 1)
1064         data1 = data[1] & 0xFF;
1065     else
1066         data1 = 0;
1067     if (data.length > 2)
1068         data2 = data[2] & 0xFF;
1069     else
1070         data2 = 0;
1071
1072     processMessage(ch, cmd, data1, data2);
1073
1074 }
1075
1076 public void processMessage(int ch, int cmd, int data1, int data2) {
1077     synchronized (synth.control_mutex) {
1078         activity();
1079     }
1080
1081     if (cmd == 0xF0) {
1082         int status = cmd | ch;
1083         switch (status) {
1084             case ShortMessage.ACTIVE_SENSING:
1085                 synchronized (synth.control_mutex) {
1086                     active_sensing_on = true;
1087                 }
1088                 break;
1089             default:
1090                 break;
1091         }
1092         return;
1093     }
1094
1095     SoftChannel[] channels = synth.channels;
1096     if (ch >= channels.length)
1097         return;
1098     SoftChannel softchannel = channels[ch];
1099
1100     switch (cmd) {
1101         case ShortMessage.NOTE_ON:
1102             if (delay_midievent != 0)
1103                 softchannel.noteOn(data1, data2, delay_midievent);
1104             else
1105                 softchannel.noteOn(data1, data2);
1106             break;
1107         case ShortMessage.NOTE_OFF:
1108             softchannel.noteOff(data1, data2);
1109             break;
1110         case ShortMessage.POLY_PRESSURE:
1111             softchannel.setPolyPressure(data1, data2);
1112             break;
1113         case ShortMessage.CONTROL_CHANGE:

```

```

1115         softchannel.controlChange(data1, data2);
1116         break;
1117     case ShortMessage.PROGRAM_CHANGE:
1118         softchannel.programChange(data1);
1119         break;
1120     case ShortMessage.CHANNEL_PRESSURE:
1121         softchannel.setChannelPressure(data1);
1122         break;
1123     case ShortMessage.PITCH_BEND:
1124         softchannel.setPitchBend(data1 + data2 * 128);
1125         break;
1126     default:
1127         break;
1128 }
1129
1130 }
1131
1132 public long getMicrosecondPosition() {
1133     if(pusher_silent)
1134     {
1135         if(synth.weakstream != null)
1136         {
1137             return (long)((sample_pos + synth.weakstream.silent_samples)
1138                 * (1000000.0 / samplerate));
1139         }
1140     }
1141     return (long)(sample_pos * (1000000.0 / samplerate));
1142 }
1143
1144 public void close() {
1145 }
1146 }

```

96 com/sun/media/sound/SoftMidiAudioFileReader.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.io.File;
28 import java.io.IOException;
29 import java.io.InputStream;
30 import java.net.URL;
31
32 import javax.sound.midi.InvalidMidiDataException;
33 import javax.sound.midi.MetaMessage;
34 import javax.sound.midi.MidiEvent;
35 import javax.sound.midi.MidiMessage;
36 import javax.sound.midi.MidiSystem;
37 import javax.sound.midi.MidiUnavailableException;
38 import javax.sound.midi.Receiver;
39 import javax.sound.midi.Sequence;
40 import javax.sound.midi.Track;
41 import javax.sound.sampled.AudioFileFormat;
42 import javax.sound.sampled.AudioFormat;
43 import javax.sound.sampled.AudioInputStream;
44 import javax.sound.sampled.UnsupportedAudioFileException;
45 import javax.sound.sampled.AudioFileFormat.Type;
46 import javax.sound.sampled.spi.AudioFileReader;
47
48 /**
49  * MIDI File Audio Renderer/Reader
50  *
51  * @author Karl Helgason
52  */
53 public class SoftMidiAudioFileReader extends AudioFileReader {
54
55     public static final Type MIDI = new Type("MIDI", "mid");
56     private static AudioFormat format = new AudioFormat(44100, 16, 2, true, false);
57
58     public AudioFileFormat getAudioFileFormat(Sequence seq)
59         throws UnsupportedAudioFileException, IOException {
60
```

```

61     long totallen = seq.getMicrosecondLength() / 1000000;
62     long len = (long) (format.getFrameRate() * (totallen + 4));
63     return new AudioFileFormat(MIDI, format, (int) len);
64 }
65
66 public AudioInputStream getAudioInputStream(Sequence seq)
67     throws UnsupportedOperationException, IOException {
68     AudioSynthesizer synth = (AudioSynthesizer) new SoftSynthesizer();
69     AudioInputStream stream;
70     Receiver recv;
71     try {
72         stream = synth.openStream(format, null);
73         recv = synth.getReceiver();
74     } catch (MidiUnavailableException e) {
75         throw new IOException(e.toString());
76     }
77     float divtype = seq.getDivisionType();
78     Track[] tracks = seq.getTracks();
79     int[] trackspos = new int[tracks.length];
80     int mpq = 500000;
81     int seqres = seq.getResolution();
82     long lasttick = 0;
83     long curtime = 0;
84     while (true) {
85         MidiEvent selevent = null;
86         int seltrack = -1;
87         for (int i = 0; i < tracks.length; i++) {
88             int trackpos = trackspos[i];
89             Track track = tracks[i];
90             if (trackpos < track.size()) {
91                 MidiEvent event = track.get(trackpos);
92                 if (selevent == null || event.getTick() < selevent.getTick()) {
93                     selevent = event;
94                     seltrack = i;
95                 }
96             }
97         }
98         if (seltrack == -1)
99             break;
100        trackspos[seltrack]++;
101        long tick = selevent.getTick();
102        if (divtype == Sequence.PPQ)
103            curtime += ((tick - lasttick) * mpq) / seqres;
104        else
105            curtime = (long) ((tick * 1000000.0 * divtype) / seqres);
106        lasttick = tick;
107        MidiMessage msg = selevent.getMessage();
108        if (msg instanceof MetaMessage) {
109            if (divtype == Sequence.PPQ) {
110                if (((MetaMessage) msg).getType() == 0x51) {
111                    byte[] data = ((MetaMessage) msg).getData();
112                    mpq = ((data[0] & 0xff) << 16)
113                        | ((data[1] & 0xff) << 8) | (data[2] & 0xff);
114                }
115            }
116        } else {
117            recv.send(msg, curtime);
118        }
119    }
120
121    long totallen = curtime / 1000000;
122    long len = (long) (stream.getFormat().getFrameRate() * (totallen + 4));

```

```

123     stream = new AudioInputStream(stream, stream.getFormat(), len);
124     return stream;
125 }
126
127 public AudioInputStream getAudioInputStream(InputStream inputstream)
128     throws UnsupportedAudioFileException, IOException {
129
130     inputstream.mark(200);
131     Sequence seq;
132     try {
133         seq = MidiSystem.getSequence(inputstream);
134     } catch (InvalidMidiDataException e) {
135         inputstream.reset();
136         throw new UnsupportedAudioFileException();
137     } catch (IOException e) {
138         inputstream.reset();
139         throw new UnsupportedAudioFileException();
140     }
141     return getAudioInputStream(seq);
142 }
143
144 public AudioFileFormat getAudioFileFormat(URL url)
145     throws UnsupportedAudioFileException, IOException {
146     Sequence seq;
147     try {
148         seq = MidiSystem.getSequence(url);
149     } catch (InvalidMidiDataException e) {
150         throw new UnsupportedAudioFileException();
151     } catch (IOException e) {
152         throw new UnsupportedAudioFileException();
153     }
154     return getAudioFileFormat(seq);
155 }
156
157 public AudioFileFormat getAudioFileFormat(File file)
158     throws UnsupportedAudioFileException, IOException {
159     Sequence seq;
160     try {
161         seq = MidiSystem.getSequence(file);
162     } catch (InvalidMidiDataException e) {
163         throw new UnsupportedAudioFileException();
164     } catch (IOException e) {
165         throw new UnsupportedAudioFileException();
166     }
167     return getAudioFileFormat(seq);
168 }
169
170 public AudioInputStream getAudioInputStream(URL url)
171     throws UnsupportedAudioFileException, IOException {
172     Sequence seq;
173     try {
174         seq = MidiSystem.getSequence(url);
175     } catch (InvalidMidiDataException e) {
176         throw new UnsupportedAudioFileException();
177     } catch (IOException e) {
178         throw new UnsupportedAudioFileException();
179     }
180     return getAudioInputStream(seq);
181 }
182
183 public AudioInputStream getAudioInputStream(File file)
184     throws UnsupportedAudioFileException, IOException {

```



```

185     if (!file.getName().toLowerCase().endsWith(".mid"))
186         throw new UnsupportedOperationException();
187     Sequence seq;
188     try {
189         seq = MidiSystem.getSequence(file);
190     } catch (InvalidMidiDataException e) {
191         throw new UnsupportedOperationException();
192     } catch (IOException e) {
193         throw new UnsupportedOperationException();
194     }
195     return getAudioInputStream(seq);
196 }
197
198 public AudioFileFormat getAudioFileFormat(InputStream inputstream)
199     throws UnsupportedOperationException, IOException {
200
201     inputstream.mark(200);
202     Sequence seq;
203     try {
204         seq = MidiSystem.getSequence(inputstream);
205     } catch (InvalidMidiDataException e) {
206         inputstream.reset();
207         throw new UnsupportedOperationException();
208     } catch (IOException e) {
209         inputstream.reset();
210         throw new UnsupportedOperationException();
211     }
212     return getAudioFileFormat(seq);
213 }
214 }

```

97 com/sun/media/sound/SoftMixingClip.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.ByteArrayOutputStream;
28 import java.io.IOException;
29 import java.io.InputStream;
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.Clip;
36 import javax.sound.sampled.DataLine;
37 import javax.sound.sampled.LineEvent;
38 import javax.sound.sampled.LineUnavailableException;
39
40 /**
41  * Clip implementation for the SoftMixingMixer.
42  *
43  * @author Karl Helgason
44  */
45 public class SoftMixingClip extends SoftMixingDataLine implements Clip {
46
47     private AudioFormat format;
48
49     private int framesize;
50
51     private byte[] data;
52
53     private InputStream datastream = new InputStream() {
54
55         public int read() throws IOException {
56             byte[] b = new byte[1];
57             int ret = read(b);
58             if (ret < 0)
59                 return ret;
60             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;

```

```

185         frameposition_sg = false;
186         afis = null;
187     } else {
188         frameposition = _frameposition;
189     }
190     if (loop_sg) {
191         _loopcount = loopcount;
192         _loopstart = loopstart;
193         _loopend = loopend;
194     }
195
196     if (afis == null) {
197         afis = AudioFloatInputStream.getInputStream(new AudioInputStream(
198             datastream, format, AudioSystem.NOT_SPECIFIED));
199
200         if (Math.abs(format.getSampleRate() - outputformat.getSampleRate()) > 0.000001)
201             afis = new AudioFloatInputStreamResampler(afis, outputformat);
202     }
203
204 }
205
206 protected void processAudioLogic(SoftAudioBuffer[] buffers) {
207     if (_active) {
208         float[] left = buffers[SoftMixingMainMixer.CHANNEL_LEFT].array();
209         float[] right = buffers[SoftMixingMainMixer.CHANNEL_RIGHT].array();
210         int bufferlen = buffers[SoftMixingMainMixer.CHANNEL_LEFT].getSize();
211
212         int readlen = bufferlen * in_nrofchannels;
213         if (readbuffer == null || readbuffer.length < readlen) {
214             readbuffer = new float[readlen];
215         }
216         int ret = 0;
217         try {
218             ret = afis.read(readbuffer);
219             if (ret == -1) {
220                 _active = false;
221                 return;
222             }
223             if (ret != in_nrofchannels)
224                 Arrays.fill(readbuffer, ret, readlen, 0);
225         } catch (IOException e) {
226         }
227
228         int in_c = in_nrofchannels;
229         for (int i = 0, ix = 0; i < bufferlen; i++, ix += in_c) {
230             left[i] += readbuffer[ix] * _leftgain;
231         }
232
233         if (out_nrofchannels != 1) {
234             if (in_nrofchannels == 1) {
235                 for (int i = 0, ix = 0; i < bufferlen; i++, ix += in_c) {
236                     right[i] += readbuffer[ix] * _rightgain;
237                 }
238             } else {
239                 for (int i = 0, ix = 1; i < bufferlen; i++, ix += in_c) {
240                     right[i] += readbuffer[ix] * _rightgain;
241                 }
242             }
243         }
244     }
245
246     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_")

```

```

309         + getFormat() + "_and_frame_length_of_" + getFrameLength());
310     }
311     if (AudioFloatConverter.getConverter(stream.getFormat()) == null)
312         throw new IllegalArgumentException("Invalid_format:_"
313             + stream.getFormat().toString());
314
315     if (stream.getFrameLength() != AudioSystem.NOT_SPECIFIED) {
316         byte[] data = new byte[(int) stream.getFrameLength()
317             * stream.getFormat().getFrameSize()];
318         int readsize = 512 * stream.getFormat().getFrameSize();
319         int len = 0;
320         while (len != data.length) {
321             if (readsize > data.length - len)
322                 readsize = data.length - len;
323             int ret = stream.read(data, len, readsize);
324             if (ret == -1)
325                 break;
326             if (ret == 0)
327                 Thread.yield();
328             len += ret;
329         }
330         open(stream.getFormat(), data, 0, len);
331     } else {
332         ByteArrayOutputStream baos = new ByteArrayOutputStream();
333         byte[] b = new byte[512 * stream.getFormat().getFrameSize()];
334         int r = 0;
335         while ((r = stream.read(b)) != -1) {
336             if (r == 0)
337                 Thread.yield();
338             baos.write(b, 0, r);
339         }
340         open(stream.getFormat(), baos.toByteArray(), 0, baos.size());
341     }
342 }
343
344
345 public void open(AudioFormat format, byte[] data, int offset, int bufferSize)
346     throws LineUnavailableException {
347     synchronized (control_mutex) {
348         if (isOpen()) {
349             throw new IllegalStateException(
350                 "Clip_is_already_open_with_format_" + getFormat()
351                 + "_and_frame_length_of_" + getFrameLength());
352         }
353         if (AudioFloatConverter.getConverter(format) == null)
354             throw new IllegalArgumentException("Invalid_format:_"
355                 + format.toString());
356         if (bufferSize % format.getFrameSize() != 0)
357             throw new IllegalArgumentException(
358                 "Buffer_size_does_not_represent_an_integral_number_of_sample_frames!");
359
360         this.data = data;
361         this.offset = offset;
362         this.bufferSize = bufferSize;
363         this.format = format;
364         this.framesize = format.getFrameSize();
365
366         loopstart = 0;
367         loopend = -1;
368         loop_sg = true;
369
370         if (!mixer.isOpen()) {

```

```

371         mixer.open();
372         mixer.implicitOpen = true;
373     }
374
375     outputformat = mixer.getFormat();
376     out_nrofchannels = outputformat.getChannels();
377     in_nrofchannels = format.getChannels();
378
379     open = true;
380
381     mixer.getMainMixer().openLine(this);
382 }
383
384 }
385
386 public void setFramePosition(int frames) {
387     synchronized (control_mutex) {
388         frameposition_sg = true;
389         frameposition = frames;
390     }
391 }
392
393 public void setLoopPoints(int start, int end) {
394     synchronized (control_mutex) {
395         if (end != -1) {
396             if (end < start)
397                 throw new IllegalArgumentException("Invalid_loop_points:_:"
398                     + start + "_-" + end);
399             if (end * framesize > bufferSize)
400                 throw new IllegalArgumentException("Invalid_loop_points:_:"
401                     + start + "_-" + end);
402         }
403         if (start * framesize > bufferSize)
404             throw new IllegalArgumentException("Invalid_loop_points:_:"
405                 + start + "_-" + end);
406         if (0 < start)
407             throw new IllegalArgumentException("Invalid_loop_points:_:"
408                 + start + "_-" + end);
409         loopstart = start;
410         loopend = end;
411         loop_sg = true;
412     }
413 }
414
415 public void setMicrosecondPosition(long microseconds) {
416     setFramePosition(((int) (microseconds * (((double) getFormat()
417         .getSampleRate()) / 1000000.0))));
418 }
419
420 public int available() {
421     return 0;
422 }
423
424 public void drain() {
425 }
426
427 public void flush() {
428 }
429
430 public int getBufferSize() {
431     return bufferSize;
432 }

```



```

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)

```

```

495         return;
496         active = false;
497         active_sg = true;
498         event = new LineEvent(this, LineEvent.Type.STOP,
499                               getLongFramePosition());
500     }
501 }
502
503 if (event != null)
504     sendEvent(event);
505 }
506
507 public void close() {
508     LineEvent event = null;
509
510     synchronized (control_mutex) {
511         if (!isOpen())
512             return;
513         stop();
514
515         event = new LineEvent(this, LineEvent.Type.CLOSE,
516                               getLongFramePosition());
517
518         open = false;
519         mixer.getMainMixer().closeLine(this);
520     }
521
522     if (event != null)
523         sendEvent(event);
524 }
525
526 public boolean isOpen() {
527     return open;
528 }
529
530 public void open() throws LineUnavailableException {
531     if (data == null) {
532         throw new IllegalArgumentException(
533             "Illegal_call_to_open()_in_interface_Clip");
534     }
535     open(format, data, offset, bufferSize);
536 }
537
538 }
539 }

```

98 com/sun/media/sound/SoftMixingDataLine.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.util.ArrayList;
29 import java.util.Arrays;
30 import java.util.List;
31
32 import javax.sound.sampled.AudioFormat;
33 import javax.sound.sampled.AudioSystem;
34 import javax.sound.sampled.BooleanControl;
35 import javax.sound.sampled.Control;
36 import javax.sound.sampled.DataLine;
37 import javax.sound.sampled.FloatControl;
38 import javax.sound.sampled.LineEvent;
39 import javax.sound.sampled.LineListener;
40 import javax.sound.sampled.Control.Type;
41
42 /**
43  * General software mixing line.
44  *
45  * @author Karl Helgason
46  */
47 public abstract class SoftMixingDataLine implements DataLine {
48
49     public static final FloatControl.Type CHORUS_SEND = new FloatControl.Type(
50         "Chorus_Send") {
51     };
52
53     protected static class AudioFloatInputStreamResampler extends
54         AudioFloatInputStream {
55
56         private AudioFloatInputStream ais;
57
58         private AudioFormat targetFormat;
59
60         private float[] skipbuffer;
```

```

61     private SoftAbstractResampler resampler;
62
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);

```

```

185     if (ibuffer_len >= 0) {
186         while (ibuffer_len < ibuffer2.length) {
187             int ret = ais.read(ibuffer2, ibuffer_len, ibuffer2.length
188                               - ibuffer_len);
189             if (ret == -1)
190                 break;
191             ibuffer_len += ret;
192         }
193         Arrays.fill(ibuffer2, ibuffer_len, ibuffer2.length, 0);
194         ibuffer_len /= nrofchannels;
195     } else {
196         Arrays.fill(ibuffer2, 0, ibuffer2.length, 0);
197     }
198
199     int ibuffer2_len = ibuffer2.length;
200     for (int c = 0; c < nrofchannels; c++) {
201         float[] buff = ibuffer[c];
202         for (int i = c, ix = pad2; i < ibuffer2_len; i += nrofchannels, ix++) {
203             buff[ix] = ibuffer2[i];
204         }
205     }
206
207 }
208
209 public int read(float[] b, int off, int len) throws IOException {
210
211     if (cbuffer == null || cbuffer[0].length < len / nrofchannels) {
212         cbuffer = new float[nrofchannels][len / nrofchannels];
213     }
214     if (ibuffer_len == -1)
215         return -1;
216     if (len < 0)
217         return 0;
218     int remain = len / nrofchannels;
219     int destPos = 0;
220     int in_end = ibuffer_len;
221     while (remain > 0) {
222         if (ibuffer_len >= 0) {
223             if (ibuffer_index >= (ibuffer_len + pad))
224                 readNextBuffer();
225             in_end = ibuffer_len + pad;
226         }
227
228         if (ibuffer_len < 0) {
229             in_end = pad2;
230             if (ibuffer_index >= in_end)
231                 break;
232         }
233
234         if (ibuffer_index < 0)
235             break;
236         int preDestPos = destPos;
237         for (int c = 0; c < nrofchannels; c++) {
238             ix[0] = ibuffer_index;
239             ox[0] = destPos;
240             float[] buff = ibuffer[c];
241             resampler.interpolate(buff, ix, in_end, pitch, 0,
242                                 cbuffer[c], ox, len / nrofchannels);
243         }
244         ibuffer_index = ix[0];
245         destPos = ox[0];
246         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     }

```

```

309     }
310
311     private class Mute extends BooleanControl {
312
313         private Mute() {
314             super(BooleanControl.Type.MUTE, false, "True", "False");
315         }
316
317         public void setValue(boolean newValue) {
318             super.setValue(newValue);
319             calcVolume();
320         }
321     }
322
323     private class ApplyReverb extends BooleanControl {
324
325         private ApplyReverb() {
326             super(BooleanControl.Type.APPLY_REVERB, false, "True", "False");
327         }
328
329         public void setValue(boolean newValue) {
330             super.setValue(newValue);
331             calcVolume();
332         }
333     }
334
335     private class Balance extends FloatControl {
336
337         private Balance() {
338             super(FloatControl.Type.BALANCE, -1.0f, 1.0f, (1.0f / 128.0f), -1,
339                 0.0f, "", "Left", "Center", "Right");
340         }
341
342         public void setValue(float newValue) {
343             super.setValue(newValue);
344             calcVolume();
345         }
346     }
347
348     private class Pan extends FloatControl {
349
350         private Pan() {
351             super(FloatControl.Type.PAN, -1.0f, 1.0f, (1.0f / 128.0f), -1,
352                 0.0f, "", "Left", "Center", "Right");
353         }
354
355         public void setValue(float newValue) {
356             super.setValue(newValue);
357             balance_control.setValue(newValue);
358         }
359
360         public float getValue() {
361             return balance_control.getValue();
362         }
363     }
364
365     private class ReverbSend extends FloatControl {
366
367         private ReverbSend() {

```



```

371         super(FloatControl.Type.REVERB_SEND, -80f, 6.0206f, 80f / 128.0f,
372               -1, -80f, "dB", "Minimum", "", "Maximum");
373     }
374
375     public void setValue(float newValue) {
376         super.setValue(newValue);
377         balance_control.setValue(newValue);
378     }
379
380 }
381
382 private class ChorusSend extends FloatControl {
383
384     private ChorusSend() {
385         super(CHORUS_SEND, -80f, 6.0206f, 80f / 128.0f, -1, -80f, "dB",
386               "Minimum", "", "Maximum");
387     }
388
389     public void setValue(float newValue) {
390         super.setValue(newValue);
391         balance_control.setValue(newValue);
392     }
393
394 }
395
396 private Gain gain_control = new Gain();
397
398 private Mute mute_control = new Mute();
399
400 private Balance balance_control = new Balance();
401
402 private Pan pan_control = new Pan();
403
404 private ReverbSend reverbsend_control = new ReverbSend();
405
406 private ChorusSend chorussend_control = new ChorusSend();
407
408 private ApplyReverb apply_reverb = new ApplyReverb();
409
410 private Control[] controls;
411
412 protected float leftgain = 1;
413
414 protected float rightgain = 1;
415
416 protected float eff1gain = 0;
417
418 protected float eff2gain = 0;
419
420 protected List<LineListener> listeners = new ArrayList<LineListener>();
421
422 protected Object control_mutex;
423
424 protected SoftMixingMixer mixer;
425
426 protected DataLine.Info info;
427
428 protected abstract void processControlLogic();
429
430 protected abstract void processAudioLogic(SoftAudioBuffer[] buffers);
431
432 protected SoftMixingDataLine(SoftMixingMixer mixer, DataLine.Info info) {

```

```

433     this.mixer = mixer;
434     this.info = info;
435     this.control_mutex = mixer.control_mutex;
436
437     controls = new Control[] { gain_control, mute_control, balance_control,
438                               pan_control, reverbsend_control, chorussend_control,
439                               apply_reverb };
440     calcVolume();
441 }
442
443 protected void calcVolume() {
444     synchronized (control_mutex) {
445         double gain = Math.pow(10.0, gain_control.getValue() / 20.0);
446         if (mute_control.getValue())
447             gain = 0;
448         leftgain = (float) gain;
449         rightgain = (float) gain;
450         if (mixer.getFormat().getChannels() > 1) {
451             // -1 = Left, 0 Center, 1 = Right
452             double balance = balance_control.getValue();
453             if (balance > 0)
454                 leftgain *= (1 - balance);
455             else
456                 rightgain *= (1 + balance);
457         }
458     }
459
460     eff1gain = (float) Math.pow(10.0, reverbsend_control.getValue() / 20.0);
461     eff2gain = (float) Math.pow(10.0, chorussend_control.getValue() / 20.0);
462
463     if (!apply_reverb.getValue()) {
464         eff1gain = 0;
465     }
466 }
467
468
469 protected void sendEvent(LineEvent event) {
470     if (listeners.size() == 0)
471         return;
472     LineListener[] listener_array = listeners
473         .toArray(new LineListener[listeners.size()]);
474     for (LineListener listener : listener_array) {
475         listener.update(event);
476     }
477 }
478
479 public void addLineListener(LineListener listener) {
480     synchronized (control_mutex) {
481         listeners.add(listener);
482     }
483 }
484
485 public void removeLineListener(LineListener listener) {
486     synchronized (control_mutex) {
487         listeners.add(listener);
488     }
489 }
490
491 public javax.sound.sampled.Line.Info getLineInfo() {
492     return info;
493 }
494

```

```

495 public Control getControl(Type control) {
496     if (control != null) {
497         for (int i = 0; i < controls.length; i++) {
498             if (controls[i].getType() == control) {
499                 return controls[i];
500             }
501         }
502     }
503     throw new IllegalArgumentException("Unsupported_control_type:_"
504         + control);
505 }
506
507 public Control[] getControls() {
508     return controls;
509 }
510
511 public boolean isControlSupported(Type control) {
512     if (control != null) {
513         for (int i = 0; i < controls.length; i++) {
514             if (controls[i].getType() == control) {
515                 return true;
516             }
517         }
518     }
519     return false;
520 }
521
522 }

```

99 com/sun/media/sound/SoftMixingMainMixer.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.List;
31
32 import javax.sound.sampled.AudioInputStream;
33 import javax.sound.sampled.AudioSystem;
34
35 /**
36  * Main mixer for SoftMixingMixer.
37  *
38  * @author Karl Helgason
39  */
40 public class SoftMixingMainMixer {
41
42     public final static int CHANNEL_LEFT = 0;
43
44     public final static int CHANNEL_RIGHT = 1;
45
46     public final static int CHANNEL_EFFECT1 = 2;
47
48     public final static int CHANNEL_EFFECT2 = 3;
49
50     public final static int CHANNEL_EFFECT3 = 4;
51
52     public final static int CHANNEL_EFFECT4 = 5;
53
54     public final static int CHANNEL_LEFT_DRY = 10;
55
56     public final static int CHANNEL_RIGHT_DRY = 11;
57
58     public final static int CHANNEL_SCRATCH1 = 12;
59
60     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     public SoftMixingMainMixer(SoftMixingMixer mixer) {
116         this.mixer = mixer;
117
118         nrofchannels = mixer.getFormat().getChannels();
119
120         int buffersize = (int) (mixer.getFormat().getSampleRate() / mixer
121             .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;

```

```

185     }
186
187     public int read(byte[] b, int off, int len) {
188         int bbuffer_len = bbuffer.length;
189         int offlen = off + len;
190         byte[] bbuffer = this.bbuffer;
191         while (off < offlen)
192             if (available() == 0)
193                 fillBuffer();
194             else {
195                 int bbuffer_pos = this.bbuffer_pos;
196                 while (off < offlen && bbuffer_pos < bbuffer_len)
197                     b[off++] = bbuffer[bbuffer_pos++];
198                 this.bbuffer_pos = bbuffer_pos;
199             }
200         return len;
201     }
202
203     public int read() throws IOException {
204         int ret = read(single);
205         if (ret == -1)
206             return -1;
207         return single[0] & 0xFF;
208     }
209
210     public int available() {
211         return bbuffer.length - bbuffer_pos;
212     }
213
214     public void close() {
215         SoftMixingMainMixer.this.mixer.close();
216     }
217
218 };
219
220 ais = new AudioInputStream(in, mixer.getFormat(),
221     AudioSystem.NOT_SPECIFIED);
222
223 }
224
225 public void openLine(SoftMixingDataLine line) {
226     synchronized (control_mutex) {
227         openLinesList.add(line);
228         openLines = openLinesList
229             .toArray(new SoftMixingDataLine[openLinesList.size()]);
230     }
231 }
232
233 public void closeLine(SoftMixingDataLine line) {
234     synchronized (control_mutex) {
235         openLinesList.remove(line);
236         openLines = openLinesList
237             .toArray(new SoftMixingDataLine[openLinesList.size()]);
238         if (openLines.length == 0)
239             if (mixer.implicitOpen)
240                 mixer.close();
241     }
242
243 }
244
245 public SoftMixingDataLine[] getOpenLines() {
246     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

```
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.util.ArrayList;
29 import java.util.List;
30
31 import javax.sound.sampled.AudioFormat;
32 import javax.sound.sampled.AudioInputStream;
33 import javax.sound.sampled.AudioSystem;
34 import javax.sound.sampled.Clip;
35 import javax.sound.sampled.Control;
36 import javax.sound.sampled.DataLine;
37 import javax.sound.sampled.Line;
38 import javax.sound.sampled.LineEvent;
39 import javax.sound.sampled.LineListener;
40 import javax.sound.sampled.LineUnavailableException;
41 import javax.sound.sampled.Mixer;
42 import javax.sound.sampled.SourceDataLine;
43 import javax.sound.sampled.AudioFormat.Encoding;
44 import javax.sound.sampled.Control.Type;
45
46 /**
47  * Software audio mixer
48  *
49  * @author Karl Helgason
50  */
51 public class SoftMixingMixer implements Mixer {
52
53     private static class Info extends Mixer.Info {
54         public Info() {
55             super(INFO_NAME, INFO_VENDOR, INFO_DESCRIPTION, INFO_VERSION);
56         }
57     }
58
59     protected static final String INFO_NAME = "Gervill_Sound_Mixer";
60 }
```

```

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,

```

```

123         AudioSystem.NOT_SPECIFIED, false));
124     formats.add(new AudioFormat(AudioFloatConverter.PCM_FLOAT,
125         AudioSystem.NOT_SPECIFIED, 32, channels, channels * 4,
126         AudioSystem.NOT_SPECIFIED, true));
127     formats.add(new AudioFormat(AudioFloatConverter.PCM_FLOAT,
128         AudioSystem.NOT_SPECIFIED, 64, channels, channels * 8,
129         AudioSystem.NOT_SPECIFIED, false));
130     formats.add(new AudioFormat(AudioFloatConverter.PCM_FLOAT,
131         AudioSystem.NOT_SPECIFIED, 64, channels, channels * 8,
132         AudioSystem.NOT_SPECIFIED, true));
133 }
134 AudioFormat[] formats_array = formats.toArray(new AudioFormat[formats
135     .size()]);
136 sourceLineInfo[0] = new DataLine.Info(SourceDataLine.class,
137     formats_array, AudioSystem.NOT_SPECIFIED,
138     AudioSystem.NOT_SPECIFIED);
139 sourceLineInfo[1] = new DataLine.Info(Clip.class, formats_array,
140     AudioSystem.NOT_SPECIFIED, AudioSystem.NOT_SPECIFIED);
141 }
142
143 public Line getLine(Line.Info info) throws LineUnavailableException {
144
145     if (!isLineSupported(info))
146         throw new IllegalArgumentException("Line_unsupported:_" + info);
147
148     if ((info.getLineClass() == SourceDataLine.class)) {
149         return new SoftMixingSourceDataLine(this, (DataLine.Info) info);
150     }
151     if ((info.getLineClass() == Clip.class)) {
152         return new SoftMixingClip(this, (DataLine.Info) info);
153     }
154
155     throw new IllegalArgumentException("Line_unsupported:_" + info);
156 }
157
158 public int getMaxLines(Line.Info info) {
159     if (info.getLineClass() == SourceDataLine.class)
160         return AudioSystem.NOT_SPECIFIED;
161     if (info.getLineClass() == Clip.class)
162         return AudioSystem.NOT_SPECIFIED;
163     return 0;
164 }
165
166 public javax.sound.sampled.Mixer.Info getMixerInfo() {
167     return info;
168 }
169
170 public javax.sound.sampled.Line.Info[] getSourceLineInfo() {
171     Line.Info[] localArray = new Line.Info[sourceLineInfo.length];
172     System.arraycopy(sourceLineInfo, 0, localArray, 0,
173         sourceLineInfo.length);
174     return localArray;
175 }
176
177 public javax.sound.sampled.Line.Info[] getSourceLineInfo(
178     javax.sound.sampled.Line.Info info) {
179     int i;
180     ArrayList<javax.sound.sampled.Line.Info> infos = new ArrayList<javax.sound.sampled.Line.
181         Info>();
182
183     for (i = 0; i < sourceLineInfo.length; i++) {
184         if (info.matches(sourceLineInfo[i])) {

```

```

184         infos.add(sourceLineInfo[i]);
185     }
186 }
187 return infos.toArray(new Line.Info[infos.size()]);
188 }
189
190 public Line[] getSourceLines() {
191     Line[] localLines;
192
193     synchronized (control_mutex) {
194         if (mainmixer == null)
195             return new Line[0];
196         SoftMixingDataLine[] sourceLines = mainmixer.getOpenLines();
197
198         localLines = new Line[sourceLines.length];
199
200         for (int i = 0; i < localLines.length; i++) {
201             localLines[i] = sourceLines[i];
202         }
203     }
204
205     return localLines;
206 }
207
208 public javax.sound.sampled.Line.Info[] getTargetLineInfo() {
209     return new javax.sound.sampled.Line.Info[0];
210 }
211
212 public javax.sound.sampled.Line.Info[] getTargetLineInfo(
213     javax.sound.sampled.Line.Info info) {
214     return new javax.sound.sampled.Line.Info[0];
215 }
216
217 public Line[] getTargetLines() {
218     return new Line[0];
219 }
220
221 public boolean isLineSupported(javax.sound.sampled.Line.Info info) {
222     if (info != null) {
223         for (int i = 0; i < sourceLineInfo.length; i++) {
224             if (info.matches(sourceLineInfo[i])) {
225                 return true;
226             }
227         }
228     }
229     return false;
230 }
231
232 public boolean isSynchronizationSupported(Line[] lines, boolean maintainSync) {
233     return false;
234 }
235
236 public void synchronize(Line[] lines, boolean maintainSync) {
237     throw new IllegalArgumentException(
238         "Synchronization_not_supported_by_this_mixer.");
239 }
240
241 public void unsynchronize(Line[] lines) {
242     throw new IllegalArgumentException(
243         "Synchronization_not_supported_by_this_mixer.");
244 }

```

```

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                 )
383                 if(format.getEncoding().equals(Encoding.PCM_SIGNED) ||
384                     format.getEncoding().equals(Encoding.PCM_UNSIGNED
385                     ))
386                     if(format.getSampleRate() == AudioSystem.NOT_SPECIFIED ||
387                         format.getSampleRate() == 48000.0)
388                     if(format.getSampleSizeInBits() == AudioSystem.
389                         NOT_SPECIFIED ||
390                         format.getSampleSizeInBits() == 16)
391                     {
392                         idealinfo = info;
393                         int ideal_channels = format.getChannels();
394                         boolean ideal_signed = format.getEncoding().equals(
395                             Encoding.PCM_SIGNED);
396                         float ideal_rate = format.getSampleRate();
397                         boolean ideal_endian = format.isBigEndian();
398                         int ideal_bits = format.getSampleSizeInBits();
399                         if(ideal_bits == AudioSystem.NOT_SPECIFIED)
400                             ideal_bits = 16;
401                         if(ideal_channels == AudioSystem.NOT_SPECIFIED)
402                             ideal_channels = 2;
403                         if(ideal_rate == AudioSystem.NOT_SPECIFIED)
404                             ideal_rate = 48000;
405                         idealformat = new AudioFormat(ideal_rate, ideal_bits,
406                             ideal_channels, ideal_signed, ideal_endian);
407                         break idealFound;
408                     }
409                 }
410             }
411         }
412     }
413     if(idealformat != null)
414     {
415         format = idealformat;
416         line = (SourceDataLine) defaultmixer.getLine(idealinfo);
417     }
418 }
419
420 if(line == null)
421     line = AudioSystem.getSourceDataLine(format);
422 } finally {
423     synchronized (SoftMixingMixerProvider.mutex) {
424         SoftMixingMixerProvider.lockthread = null;
425     }
426 }
427
428 if (line == null)
429     throw new IllegalArgumentException("No_line_matching_"
430         + info.toString() + "_is_supported.");
431 }

```

```

425     double latency = this.latency;
426
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

```



```

487         if (targetFormat != null)
488             format = targetFormat;
489
490         mainmixer = new SoftMixingMainMixer(this);
491
492         sendEvent(new LineEvent(this, LineEvent.Type.OPEN,
493             AudioSystem.NOT_SPECIFIED));
494
495         return mainmixer.getInputStream();
496     }
497 }
498
499 }
500
501 public void removeLineListener(LineListener listener) {
502     synchronized (control_mutex) {
503         listeners.remove(listener);
504     }
505 }
506
507 public long getLatency() {
508     synchronized (control_mutex) {
509         return latency;
510     }
511 }
512
513 public AudioFormat getFormat() {
514     synchronized (control_mutex) {
515         return format;
516     }
517 }
518
519 protected float getControlRate() {
520     return controlrate;
521 }
522
523 protected SoftMixingMainMixer getMainMixer() {
524     if (!isOpen())
525         return null;
526     return mainmixer;
527 }
528
529 }

```

101 com/sun/media/sound/SoftMixingMixerProvider.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 javax.sound.sampled.Mixer;
28  import javax.sound.sampled.Mixer.Info;
29  import javax.sound.sampled.spi.MixerProvider;
30
31  /**
32   * Provider for software audio mixer
33   *
34   * @author Karl Helgason
35   */
36  public class SoftMixingMixerProvider extends MixerProvider {
37
38      static SoftMixingMixer globalmixer = null;
39
40      static Thread lockthread = null;
41
42      protected final static Object mutex = new Object();
43
44      public Mixer getMixer(Info info) {
45          if (!(info == null || info == SoftMixingMixer.info)) {
46              throw new IllegalArgumentException("Mixer_" + info.toString()
47                  + "_not_supported_by_this_provider.");
48          }
49          synchronized (mutex) {
50              if (lockthread != null)
51                  if (Thread.currentThread() == lockthread)
52                      throw new IllegalArgumentException("Mixer_"
53                          + info.toString()
54                          + "_not_supported_by_this_provider.");
55              if (globalmixer == null)
56                  globalmixer = new SoftMixingMixer();
57              return globalmixer;
58          }
59      }
60  }
```

```
61
62     public Info[] getMixerInfo() {
63         return new Info[] { SoftMixingMixer.info };
64     }
65
66 }
```

```

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.Arrays;
30
31  import javax.sound.sampled.AudioFormat;
32  import javax.sound.sampled.AudioInputStream;
33  import javax.sound.sampled.AudioSystem;
34  import javax.sound.sampled.DataLine;
35  import javax.sound.sampled.LineEvent;
36  import javax.sound.sampled.LineUnavailableException;
37  import javax.sound.sampled.SourceDataLine;
38
39  /**
40   * SourceDataLine implementation for the SoftMixingMixer.
41   *
42   * @author Karl Helgason
43   */
44  public class SoftMixingSourceDataLine extends SoftMixingDataLine implements
45      SourceDataLine {
46
47      private boolean open = false;
48
49      private AudioFormat format = new AudioFormat(44100.0f, 16, 2, true, false);
50
51      private int framesize;
52
53      private int bufferSize = -1;
54
55      private float[] readbuffer;
56
57      private boolean active = false;
58
59      private byte[] cycling_buffer;
60

```

```

61     private int cycling_read_pos = 0;
62
63     private int cycling_write_pos = 0;
64
65     private int cycling_avail = 0;
66
67     private long cycling_framepos = 0;
68
69     private AudioFloatInputStream afis;
70
71     private static class NonBlockingFloatInputStream extends
72         AudioFloatInputStream {
73         AudioFloatInputStream ais;
74
75         public NonBlockingFloatInputStream(AudioFloatInputStream ais) {
76             this.ais = ais;
77         }
78
79         public int available() throws IOException {
80             return ais.available();
81         }
82
83         public void close() throws IOException {
84             ais.close();
85         }
86
87         public AudioFormat getFormat() {
88             return ais.getFormat();
89         }
90
91         public long getFrameLength() {
92             return ais.getFrameLength();
93         }
94
95         public void mark(int readlimit) {
96             ais.mark(readlimit);
97         }
98
99         public boolean markSupported() {
100             return ais.markSupported();
101         }
102
103         public int read(float[] b, int off, int len) throws IOException {
104             int avail = available();
105             if (len > avail) {
106                 int ret = ais.read(b, off, avail);
107                 Arrays.fill(b, off + ret, off + len, 0);
108                 return len;
109             }
110             return ais.read(b, off, len);
111         }
112
113         public void reset() throws IOException {
114             ais.reset();
115         }
116
117         public long skip(long len) throws IOException {
118             return ais.skip(len);
119         }
120     }
121 }
122

```

```

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             if (_eff1gain > 0.0001) {
242                 float[] eff1 = buffers[SoftMixingMainMixer.CHANNEL_EFFECT1]
243                     .array();
244                 for (int i = 0, ix = 0; i < bufferlen; i++, ix += in_c) {
245                     eff1[i] += readbuffer[ix] * _eff1gain;
246

```

```

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     mixer.getMainMixer().openLine(this);
310
311     cycling_buffer = new byte[framesize * bufferSize];
312     cycling_read_pos = 0;
313     cycling_write_pos = 0;
314     cycling_avail = 0;
315     cycling_framepos = 0;
316
317     InputStream cycling_inputstream = new InputStream() {
318
319         public int read() throws IOException {
320             byte[] b = new byte[1];
321             int ret = read(b);
322             if (ret < 0)
323                 return ret;
324             return b[0] & 0xFF;
325         }
326
327         public int available() throws IOException {
328             synchronized (cycling_buffer) {
329                 return cycling_avail;
330             }
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())) {

```

```

371         throw new IllegalStateException(
372             "Line_is_already_open_with_format_" + getFormat()
373             + "_and_bufferSize_" + getBufferSize());
374     }
375 }
376
377 }
378
379 if (event != null)
380     sendEvent(event);
381
382 }
383
384 public int available() {
385     synchronized (cycling_buffer) {
386         return cycling_buffer.length - cycling_avail;
387     }
388 }
389
390 public void drain() {
391     while (true) {
392         int avail;
393         synchronized (cycling_buffer) {
394             avail = cycling_avail;
395         }
396         if (avail != 0)
397             return;
398         try {
399             Thread.sleep(1);
400         } catch (InterruptedException e) {
401             return;
402         }
403     }
404 }
405
406 public void flush() {
407     synchronized (cycling_buffer) {
408         cycling_read_pos = 0;
409         cycling_write_pos = 0;
410         cycling_avail = 0;
411     }
412 }
413
414 public int getBufferSize() {
415     synchronized (control_mutex) {
416         return bufferSize;
417     }
418 }
419
420 public AudioFormat getFormat() {
421     synchronized (control_mutex) {
422         return format;
423     }
424 }
425
426 public int getFramePosition() {
427     return (int) getLongFramePosition();
428 }
429
430 public float getLevel() {
431     return AudioSystem.NOT_SPECIFIED;
432 }

```

```

433     public long getLongFramePosition() {
434         synchronized (cycling_buffer) {
435             return cycling_framepos;
436         }
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 }
```

```

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.ArrayList;
28 import java.util.Arrays;
29 import java.util.Comparator;
30 import java.util.HashMap;
31 import java.util.List;
32 import java.util.Map;
33
34 /**
35 * This class decodes information from ModelPeformer for use in SoftVoice.
36 * It also adds default connections if they where missing in ModelPerformer.
37 *
38 * @author Karl Helgason
39 */
40 public class SoftPerformer {
41
42     static ModelConnectionBlock[] defaultconnections
43         = new ModelConnectionBlock[42];
44
45     static {
46         int o = 0;
47         defaultconnections[o++] = new ModelConnectionBlock(
48             new ModelSource(
49                 new ModelIdentifier("noteon", "on", 0),
50                 ModelStandardTransform.DIRECTION_MIN2MAX,
51                 ModelStandardTransform.POLARITY_UNIPOLAR,
52                 ModelStandardTransform.TRANSFORM_LINEAR),
53             1, new ModelDestination(new ModelIdentifier("eg", "on", 0)));
54
55         defaultconnections[o++] = new ModelConnectionBlock(
56             new ModelSource(
57                 new ModelIdentifier("noteon", "on", 0),
58                 ModelStandardTransform.DIRECTION_MIN2MAX,
59                 ModelStandardTransform.POLARITY_UNIPOLAR,
60                 ModelStandardTransform.TRANSFORM_LINEAR),

```

```

61     1, new ModelDestination(new ModelIdentifier("eg", "on", 1)));
62
63 defaultconnections[o++] = new ModelConnectionBlock(
64     new ModelSource(
65         new ModelIdentifier("eg", "active", 0),
66         ModelStandardTransform.DIRECTION_MIN2MAX,
67         ModelStandardTransform.POLARITY_UNIPOLAR,
68         ModelStandardTransform.TRANSFORM_LINEAR),
69     1, new ModelDestination(new ModelIdentifier("mixer", "active", 0)));
70
71 defaultconnections[o++] = new ModelConnectionBlock(
72     new ModelSource(
73         new ModelIdentifier("eg", 0),
74         ModelStandardTransform.DIRECTION_MAX2MIN,
75         ModelStandardTransform.POLARITY_UNIPOLAR,
76         ModelStandardTransform.TRANSFORM_LINEAR),
77     -960, new ModelDestination(new ModelIdentifier("mixer", "gain")));
78
79 defaultconnections[o++] = new ModelConnectionBlock(
80     new ModelSource(
81         new ModelIdentifier("noteon", "velocity"),
82         ModelStandardTransform.DIRECTION_MAX2MIN,
83         ModelStandardTransform.POLARITY_UNIPOLAR,
84         ModelStandardTransform.TRANSFORM_CONCAVE),
85     -960, new ModelDestination(new ModelIdentifier("mixer", "gain")));
86
87 defaultconnections[o++] = new ModelConnectionBlock(
88     new ModelSource(
89         new ModelIdentifier("midi", "pitch"),
90         ModelStandardTransform.DIRECTION_MIN2MAX,
91         ModelStandardTransform.POLARITY_BIPOLAR,
92         ModelStandardTransform.TRANSFORM_LINEAR),
93     new ModelSource(new ModelIdentifier("midi_rpn", "0"),
94         new ModelTransform() {
95             public double transform(double value) {
96                 int v = (int) (value * 16384.0);
97                 int msb = v >> 7;
98                 int lsb = v & 127;
99                 return msb * 100 + lsb;
100             }
101         }),
102     new ModelDestination(new ModelIdentifier("osc", "pitch")));
103
104 defaultconnections[o++] = new ModelConnectionBlock(
105     new ModelSource(
106         new ModelIdentifier("noteon", "keynumber"),
107         ModelStandardTransform.DIRECTION_MIN2MAX,
108         ModelStandardTransform.POLARITY_UNIPOLAR,
109         ModelStandardTransform.TRANSFORM_LINEAR),
110     12800, new ModelDestination(new ModelIdentifier("osc", "pitch")));
111
112 defaultconnections[o++] = new ModelConnectionBlock(
113     new ModelSource(
114         new ModelIdentifier("midi_cc", "7"),
115         ModelStandardTransform.DIRECTION_MAX2MIN,
116         ModelStandardTransform.POLARITY_UNIPOLAR,
117         ModelStandardTransform.TRANSFORM_CONCAVE),
118     -960, new ModelDestination(new ModelIdentifier("mixer", "gain")));
119
120 defaultconnections[o++] = new ModelConnectionBlock(
121     new ModelSource(
122         new ModelIdentifier("midi_cc", "8"),

```

```

123         ModelStandardTransform.DIRECTION_MIN2MAX,
124         ModelStandardTransform.POLARITY_UNIPOLAR,
125         ModelStandardTransform.TRANSFORM_LINEAR),
126     1000, new ModelDestination(new ModelIdentifier("mixer", "balance"))));
127
128 defaultconnections[o++] = new ModelConnectionBlock(
129     new ModelSource(
130         new ModelIdentifier("midi_cc", "10"),
131         ModelStandardTransform.DIRECTION_MIN2MAX,
132         ModelStandardTransform.POLARITY_UNIPOLAR,
133         ModelStandardTransform.TRANSFORM_LINEAR),
134     1000, new ModelDestination(new ModelIdentifier("mixer", "pan"))));
135
136 defaultconnections[o++] = new ModelConnectionBlock(
137     new ModelSource(
138         new ModelIdentifier("midi_cc", "11"),
139         ModelStandardTransform.DIRECTION_MAX2MIN,
140         ModelStandardTransform.POLARITY_UNIPOLAR,
141         ModelStandardTransform.TRANSFORM_CONCAVE),
142     -960, new ModelDestination(new ModelIdentifier("mixer", "gain"))));
143
144 defaultconnections[o++] = new ModelConnectionBlock(
145     new ModelSource(
146         new ModelIdentifier("midi_cc", "91"),
147         ModelStandardTransform.DIRECTION_MIN2MAX,
148         ModelStandardTransform.POLARITY_UNIPOLAR,
149         ModelStandardTransform.TRANSFORM_LINEAR),
150     1000, new ModelDestination(new ModelIdentifier("mixer", "reverb"))));
151
152 defaultconnections[o++] = new ModelConnectionBlock(
153     new ModelSource(
154         new ModelIdentifier("midi_cc", "93"),
155         ModelStandardTransform.DIRECTION_MIN2MAX,
156         ModelStandardTransform.POLARITY_UNIPOLAR,
157         ModelStandardTransform.TRANSFORM_LINEAR),
158     1000, new ModelDestination(new ModelIdentifier("mixer", "chorus"))));
159
160 defaultconnections[o++] = new ModelConnectionBlock(
161     new ModelSource(
162         new ModelIdentifier("midi_cc", "71"),
163         ModelStandardTransform.DIRECTION_MIN2MAX,
164         ModelStandardTransform.POLARITY_BIPOLAR,
165         ModelStandardTransform.TRANSFORM_LINEAR),
166     200, new ModelDestination(new ModelIdentifier("filter", "q"))));
167 defaultconnections[o++] = new ModelConnectionBlock(
168     new ModelSource(
169         new ModelIdentifier("midi_cc", "74"),
170         ModelStandardTransform.DIRECTION_MIN2MAX,
171         ModelStandardTransform.POLARITY_BIPOLAR,
172         ModelStandardTransform.TRANSFORM_LINEAR),
173     9600, new ModelDestination(new ModelIdentifier("filter", "freq"))));
174
175 defaultconnections[o++] = new ModelConnectionBlock(
176     new ModelSource(
177         new ModelIdentifier("midi_cc", "72"),
178         ModelStandardTransform.DIRECTION_MIN2MAX,
179         ModelStandardTransform.POLARITY_BIPOLAR,
180         ModelStandardTransform.TRANSFORM_LINEAR),
181     6000, new ModelDestination(new ModelIdentifier("eg", "release2"))));
182
183 defaultconnections[o++] = new ModelConnectionBlock(
184     new ModelSource(

```

```

185         new ModelIdentifier("midi_cc", "73"),
186         ModelStandardTransform.DIRECTION_MIN2MAX,
187         ModelStandardTransform.POLARITY_BIPOLAR,
188         ModelStandardTransform.TRANSFORM_LINEAR),
189     2000, new ModelDestination(new ModelIdentifier("eg", "attack2"))));
190
191 defaultconnections[o++] = new ModelConnectionBlock(
192     new ModelSource(
193         new ModelIdentifier("midi_cc", "75"),
194         ModelStandardTransform.DIRECTION_MIN2MAX,
195         ModelStandardTransform.POLARITY_BIPOLAR,
196         ModelStandardTransform.TRANSFORM_LINEAR),
197     6000, new ModelDestination(new ModelIdentifier("eg", "decay2"))));
198
199 defaultconnections[o++] = new ModelConnectionBlock(
200     new ModelSource(
201         new ModelIdentifier("midi_cc", "67"),
202         ModelStandardTransform.DIRECTION_MIN2MAX,
203         ModelStandardTransform.POLARITY_UNIPOLAR,
204         ModelStandardTransform.TRANSFORM_SWITCH),
205     -50, new ModelDestination(ModelDestination.DESTINATION_GAIN));
206
207 defaultconnections[o++] = new ModelConnectionBlock(
208     new ModelSource(
209         new ModelIdentifier("midi_cc", "67"),
210         ModelStandardTransform.DIRECTION_MIN2MAX,
211         ModelStandardTransform.POLARITY_UNIPOLAR,
212         ModelStandardTransform.TRANSFORM_SWITCH),
213     -2400, new ModelDestination(ModelDestination.DESTINATION_FILTER_FREQ));
214
215 defaultconnections[o++] = new ModelConnectionBlock(
216     new ModelSource(
217         new ModelIdentifier("midi_rpn", "1"),
218         ModelStandardTransform.DIRECTION_MIN2MAX,
219         ModelStandardTransform.POLARITY_BIPOLAR,
220         ModelStandardTransform.TRANSFORM_LINEAR),
221     100, new ModelDestination(new ModelIdentifier("osc", "pitch"))));
222
223 defaultconnections[o++] = new ModelConnectionBlock(
224     new ModelSource(
225         new ModelIdentifier("midi_rpn", "2"),
226         ModelStandardTransform.DIRECTION_MIN2MAX,
227         ModelStandardTransform.POLARITY_BIPOLAR,
228         ModelStandardTransform.TRANSFORM_LINEAR),
229     12800, new ModelDestination(new ModelIdentifier("osc", "pitch"))));
230
231 defaultconnections[o++] = new ModelConnectionBlock(
232     new ModelSource(
233         new ModelIdentifier("master", "fine_tuning"),
234         ModelStandardTransform.DIRECTION_MIN2MAX,
235         ModelStandardTransform.POLARITY_BIPOLAR,
236         ModelStandardTransform.TRANSFORM_LINEAR),
237     100, new ModelDestination(new ModelIdentifier("osc", "pitch"))));
238
239 defaultconnections[o++] = new ModelConnectionBlock(
240     new ModelSource(
241         new ModelIdentifier("master", "coarse_tuning"),
242         ModelStandardTransform.DIRECTION_MIN2MAX,
243         ModelStandardTransform.POLARITY_BIPOLAR,
244         ModelStandardTransform.TRANSFORM_LINEAR),
245     12800, new ModelDestination(new ModelIdentifier("osc", "pitch"))));
246

```



```

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;

```

```

309 public ModelPerformer performer;
310 public ModelConnectionBlock[] connections;
311 public ModelOscillator[] oscillators;
312 public Map<Integer, int[]> midi_rpn_connections = new HashMap<Integer, int[]>();
313 public Map<Integer, int[]> midi_nrpn_connections = new HashMap<Integer, int[]>();
314 public int[][] midi_ctrl_connections;
315 public int[][] midi_connections;
316 public int[] ctrl_connections;
317 private List<Integer> ctrl_connections_list = new ArrayList<Integer>();
318
319 private static class KeySortComparator implements Comparator<ModelSource> {
320
321     public int compare(ModelSource o1, ModelSource o2) {
322         return o1.getIdentifier().toString().compareTo(
323             o2.getIdentifier().toString());
324     }
325 }
326 private static KeySortComparator keySortComparator = new KeySortComparator();
327
328 private String extractKeys(ModelConnectionBlock conn) {
329     StringBuffer sb = new StringBuffer();
330     if (conn.getSources() != null) {
331         sb.append("[");
332         ModelSource[] srcs = conn.getSources();
333         ModelSource[] srcs2 = new ModelSource[srcs.length];
334         for (int i = 0; i < srcs.length; i++)
335             srcs2[i] = srcs[i];
336         Arrays.sort(srcs2, keySortComparator);
337         for (int i = 0; i < srcs.length; i++) {
338             sb.append(srcs[i].getIdentifier());
339             sb.append(";");
340         }
341         sb.append("]");
342     }
343     sb.append(";");
344     if (conn.getDestination() != null) {
345         sb.append(conn.getDestination().getIdentifier());
346     }
347     sb.append(";");
348     return sb.toString();
349 }
350
351 private void processSource(ModelSource src, int ix) {
352     ModelIdentifier id = src.getIdentifier();
353     String o = id.getObject();
354     if (o.equals("midi_cc"))
355         processMidiControlSource(src, ix);
356     else if (o.equals("midi_rpn"))
357         processMidiRpnSource(src, ix);
358     else if (o.equals("midi_nrpn"))
359         processMidiNrpnSource(src, ix);
360     else if (o.equals("midi"))
361         processMidiSource(src, ix);
362     else if (o.equals("noteon"))
363         processNoteOnSource(src, ix);
364     else if (o.equals("osc"))
365         return;
366     else if (o.equals("mixer"))
367         return;
368     else
369         ctrl_connections_list.add(ix);
370 }

```

```

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

```

```

433 private void processMidiRpnSource(ModelSource src, int ix) {
434     String v = src.getIdentifier().getVariable();
435     if (v == null)
436         return;
437     int c = Integer.parseInt(v);
438     if (midi_rpn_connections.get(c) == null)
439         midi_rpn_connections.put(c, new int[]{ix});
440     else {
441         int[] olda = midi_rpn_connections.get(c);
442         int[] newa = new int[olda.length + 1];
443         for (int i = 0; i < olda.length; i++)
444             newa[i] = olda[i];
445         newa[newa.length - 1] = ix;
446         midi_rpn_connections.put(c, newa);
447     }
448 }
449
450 private void processMidiNrpnSource(ModelSource src, int ix) {
451     String v = src.getIdentifier().getVariable();
452     if (v == null)
453         return;
454     int c = Integer.parseInt(v);
455     if (midi_nrpn_connections.get(c) == null)
456         midi_nrpn_connections.put(c, new int[]{ix});
457     else {
458         int[] olda = midi_nrpn_connections.get(c);
459         int[] newa = new int[olda.length + 1];
460         for (int i = 0; i < olda.length; i++)
461             newa[i] = olda[i];
462         newa[newa.length - 1] = ix;
463         midi_nrpn_connections.put(c, newa);
464     }
465 }
466
467 public SoftPerformer(ModelPerformer performer) {
468     this.performer = performer;
469
470     keyFrom = performer.getKeyFrom();
471     keyTo = performer.getKeyTo();
472     velFrom = performer.getVelFrom();
473     velTo = performer.getVelTo();
474     exclusiveClass = performer.getExclusiveClass();
475     selfNonExclusive = performer.isSelfNonExclusive();
476
477     Map<String, ModelConnectionBlock> connmap = new HashMap<String, ModelConnectionBlock>();
478
479     List<ModelConnectionBlock> performer_connections = new ArrayList<ModelConnectionBlock>();
480     performer_connections.addAll(performer.getConnectionBlocks());
481
482     if (performer.isDefaultConnectionsEnabled()) {
483
484         // Add modulation depth range (RPN 5) to the modulation wheel (cc#1)
485
486         boolean isModulationWheelConectionFound = false;
487         for (int j = 0; j < performer_connections.size(); j++) {
488             ModelConnectionBlock connection = performer_connections.get(j);
489             ModelSource[] sources = connection.getSources();
490             ModelDestination dest = connection.getDestination();
491             boolean isModulationWheelConection = false;
492             if (dest != null && sources != null && sources.length > 1) {
493                 for (int i = 0; i < sources.length; i++) {
494                     // check if connection block has the source "modulation

```

```

495         // wheel cc#1"
496         if (sources[i].getIdentifier().getObject().equals(
497             "midi_cc")) {
498             if (sources[i].getIdentifier().getVariable()
499                 .equals("1")) {
500                 isModulationWheelConection = true;
501                 isModulationWheelConectionFound = true;
502                 break;
503             }
504         }
505     }
506 }
507 if (isModulationWheelConection) {
508
509     ModelConnectionBlock newconnection = new ModelConnectionBlock();
510     newconnection.setSources(connection.getSources());
511     newconnection.setDestination(connection.getDestination());
512     newconnection.addSource(new ModelSource(
513         new ModelIdentifier("midi_rpn", "5")));
514     newconnection.setScale(connection.getScale() * 256.0);
515     performer_connections.set(j, newconnection);
516 }
517 }
518
519 if (!isModulationWheelConectionFound) {
520     ModelConnectionBlock conn = new ModelConnectionBlock(
521         new ModelSource(ModelSource.SOURCE_LF01,
522             ModelStandardTransform.DIRECTION_MIN2MAX,
523             ModelStandardTransform.POLARITY_BIPOLAR,
524             ModelStandardTransform.TRANSFORM_LINEAR),
525         new ModelSource(new ModelIdentifier("midi_cc", "1", 0),
526             ModelStandardTransform.DIRECTION_MIN2MAX,
527             ModelStandardTransform.POLARITY_UNIPOLAR,
528             ModelStandardTransform.TRANSFORM_LINEAR),
529         50,
530         new ModelDestination(ModelDestination.DESTINATION_PITCH));
531     conn.addSource(new ModelSource(new ModelIdentifier("midi_rpn",
532         "5"))));
533     conn.setScale(conn.getScale() * 256.0);
534     performer_connections.add(conn);
535 }
536
537 // Let Aftertouch to behave just like modulation wheel (cc#1)
538 boolean channel_pressure_set = false;
539 boolean poly_pressure = false;
540 ModelConnectionBlock mod_cc_1_connection = null;
541 int mod_cc_1_connection_src_ix = 0;
542
543 for (ModelConnectionBlock connection : performer_connections) {
544     ModelSource[] sources = connection.getSources();
545     ModelDestination dest = connection.getDestination();
546     // if(dest != null && sources != null)
547     if (dest != null && sources != null) {
548         for (int i = 0; i < sources.length; i++) {
549             ModelIdentifier srcid = sources[i].getIdentifier();
550             // check if connection block has the source "modulation
551             // wheel cc#1"
552             if (srcid.getObject().equals("midi_cc")) {
553                 if (srcid.getVariable().equals("1")) {
554                     mod_cc_1_connection = connection;
555                     mod_cc_1_connection_src_ix = i;

```

```

557     }
558 }
559 // check if channel or poly pressure are already
560 // connected
561 if (srcid.getObject().equals("midi")) {
562     if (srcid.getVariable().equals("channel_pressure"))
563         channel_pressure_set = true;
564     if (srcid.getVariable().equals("poly_pressure"))
565         poly_pressure = true;
566 }
567 }
568 }
569
570 }
571
572 if (mod_cc_1_connection != null) {
573     if (!channel_pressure_set) {
574         ModelConnectionBlock mc = new ModelConnectionBlock();
575         mc.setDestination(mod_cc_1_connection.getDestination());
576         mc.setScale(mod_cc_1_connection.getScale());
577         ModelSource[] src_list = mod_cc_1_connection.getSources();
578         ModelSource[] src_list_new = new ModelSource[src_list.length];
579         for (int i = 0; i < src_list_new.length; i++)
580             src_list_new[i] = src_list[i];
581         src_list_new[mod_cc_1_connection_src_ix] = new ModelSource(
582             new ModelIdentifier("midi", "channel_pressure"));
583         mc.setSources(src_list_new);
584         connmap.put(extractKeys(mc), mc);
585     }
586     if (!poly_pressure) {
587         ModelConnectionBlock mc = new ModelConnectionBlock();
588         mc.setDestination(mod_cc_1_connection.getDestination());
589         mc.setScale(mod_cc_1_connection.getScale());
590         ModelSource[] src_list = mod_cc_1_connection.getSources();
591         ModelSource[] src_list_new = new ModelSource[src_list.length];
592         for (int i = 0; i < src_list_new.length; i++)
593             src_list_new[i] = src_list[i];
594         src_list_new[mod_cc_1_connection_src_ix] = new ModelSource(
595             new ModelIdentifier("midi", "poly_pressure"));
596         mc.setSources(src_list_new);
597         connmap.put(extractKeys(mc), mc);
598     }
599 }
600
601 // Enable Vibration Sound Controllers : 76, 77, 78
602 ModelConnectionBlock found_vib_connection = null;
603 for (ModelConnectionBlock connection : performer_connections) {
604     ModelSource[] sources = connection.getSources();
605     if (sources.length != 0
606         && sources[0].getIdentifier().getObject().equals("lfo")) {
607         if (connection.getDestination().getIdentifier().equals(
608             ModelDestination.DESTINATION_PITCH)) {
609             if (found_vib_connection == null)
610                 found_vib_connection = connection;
611             else {
612                 if (found_vib_connection.getSources().length > sources.length)
613                     found_vib_connection = connection;
614                 else if (found_vib_connection.getSources()[0]
615                     .getIdentifier().getInstance() < 1) {
616                     if (found_vib_connection.getSources()[0]
617                         .getIdentifier().getInstance() >
618                         sources[0].getIdentifier().getInstance()) {

```

```

619         found_vib_connection = connection;
620     }
621 }
622 }
623
624 }
625 }
626 }
627
628 int instance = 1;
629
630 if (found_vib_connection != null) {
631     instance = found_vib_connection.getSources()[0].getIdentifier()
632         .getInstance();
633 }
634 ModelConnectionBlock connection;
635
636 connection = new ModelConnectionBlock(
637     new ModelSource(new ModelIdentifier("midi_cc", "78"),
638         ModelStandardTransform.DIRECTION_MIN2MAX,
639         ModelStandardTransform.POLARITY_BIPOLAR,
640         ModelStandardTransform.TRANSFORM_LINEAR),
641     2000, new ModelDestination(
642         new ModelIdentifier("lfo", "delay2", instance)));
643 connmap.put(extractKeys(connection), connection);
644
645 final double scale = found_vib_connection == null ? 0
646     : found_vib_connection.getScale();
647 connection = new ModelConnectionBlock(
648     new ModelSource(new ModelIdentifier("lfo", instance)),
649     new ModelSource(new ModelIdentifier("midi_cc", "77"),
650         new ModelTransform() {
651             double s = scale;
652             public double transform(double value) {
653                 value = value * 2 - 1;
654                 value *= 600;
655                 if (s == 0) {
656                     return value;
657                 } else if (s > 0) {
658                     if (value < -s)
659                         value = -s;
660                     return value;
661                 } else {
662                     if (value < s)
663                         value = -s;
664                     return -value;
665                 }
666             }
667         }, new ModelDestination(ModelDestination.DESTINATION_PITCH));
668 connmap.put(extractKeys(connection), connection);
669
670 connection = new ModelConnectionBlock(
671     new ModelSource(new ModelIdentifier("midi_cc", "76"),
672         ModelStandardTransform.DIRECTION_MIN2MAX,
673         ModelStandardTransform.POLARITY_BIPOLAR,
674         ModelStandardTransform.TRANSFORM_LINEAR),
675     2400, new ModelDestination(
676         new ModelIdentifier("lfo", "freq", instance)));
677 connmap.put(extractKeys(connection), connection);
678
679 }
680

```

```

681 // Add default connection blocks
682 if (performer.isDefaultConnectionsEnabled())
683     for (ModelConnectionBlock connection : defaultconnections)
684         connmap.put(extractKeys(connection), connection);
685 // Add connection blocks from modelperformer
686 for (ModelConnectionBlock connection : performer_connections)
687     connmap.put(extractKeys(connection), connection);
688 // separeate connection blocks : Init time, Midi Time, Midi/Control Time,
689 // Control Time
690 List<ModelConnectionBlock> connections = new ArrayList<ModelConnectionBlock>();
691
692 midi_ctrl_connections = new int[128][];
693 for (int i = 0; i < midi_ctrl_connections.length; i++) {
694     midi_ctrl_connections[i] = null;
695 }
696 midi_connections = new int[5][];
697 for (int i = 0; i < midi_connections.length; i++) {
698     midi_connections[i] = null;
699 }
700
701 int ix = 0;
702 boolean mustBeOnTop = false;
703
704 for (ModelConnectionBlock connection : connmap.values()) {
705     if (connection.getDestination() != null) {
706         ModelDestination dest = connection.getDestination();
707         ModelIdentifier id = dest.getIdentifer();
708         if (id.getObject().equals("noteon")) {
709             mustBeOnTop = true;
710             if (id.getVariable().equals("keynumber"))
711                 forcedKeynumber = true;
712             if (id.getVariable().equals("velocity"))
713                 forcedVelocity = true;
714         }
715     }
716     if (mustBeOnTop) {
717         connections.add(0, connection);
718         mustBeOnTop = false;
719     } else
720         connections.add(connection);
721 }
722
723 for (ModelConnectionBlock connection : connections) {
724     if (connection.getSources() != null) {
725         ModelSource[] srcs = connection.getSources();
726         for (int i = 0; i < srcs.length; i++) {
727             processSource(srcs[i], ix);
728         }
729     }
730     ix++;
731 }
732
733 this.connections = new ModelConnectionBlock[connections.size()];
734 connections.toArray(this.connections);
735
736 this.ctrl_connections = new int[ctrl_connections_list.size()];
737
738 for (int i = 0; i < this.ctrl_connections.length; i++)
739     this.ctrl_connections[i] = ctrl_connections_list.get(i);
740
741 oscillators = new ModelOscillator[performer.getOscillators().size()];
742 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 private static boolean isUnnecessaryTransform(ModelTransform transform) {
761     if (transform == null)
762         return false;
763     if (!(transform instanceof ModelStandardTransform))
764         return false;
765     ModelStandardTransform stransform = (ModelStandardTransform)transform;
766     if (stransform.getDirection() != ModelStandardTransform.DIRECTION_MIN2MAX)
767         return false;
768     if (stransform.getPolarity() != ModelStandardTransform.POLARITY_UNIPOLAR)
769         return false;
770     if (stransform.getTransform() != ModelStandardTransform.TRANSFORM_LINEAR)
771         return false;
772     return false;
773 }
774 }
775 }

```

```

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 /**
28 * A resampler that uses 0-order (nearest-neighbor) interpolation.
29 *
30 * @author Karl Helgason
31 */
32 public class SoftPointResampler extends SoftAbstractResampler {
33
34     public int getPadding() {
35         return 100;
36     }
37
38     public void interpolate(float[] in, float[] in_offset, float in_end,
39         float[] startpitch, float pitchstep, float[] out, int[] out_offset,
40         int out_end) {
41         float pitch = startpitch[0];
42         float ix = in_offset[0];
43         int ox = out_offset[0];
44         float ix_end = in_end;
45         float ox_end = out_end;
46         if (pitchstep == 0) {
47             while (ix < ix_end && ox < ox_end) {
48                 out[ox++] = in[(int) ix];
49                 ix += pitch;
50             }
51         } else {
52             while (ix < ix_end && ox < ox_end) {
53                 out[ox++] = in[(int) ix];
54                 ix += pitch;
55                 pitch += pitchstep;
56             }
57         }
58         in_offset[0] = ix;
59         out_offset[0] = ox;
60         startpitch[0] = pitch;

```

61
62 }
63 }

105 com/sun/media/sound/SoftProcess.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 /**
28 * Control signal processor interface
29 *
30 * @author Karl Helgason
31 */
32 public interface SoftProcess extends SoftControl {
33
34     public void init(SoftSynthesizer synth);
35
36     public double[] get(int instance, String name);
37
38     public void processControlLogic();
39
40     public void reset();
41 }
```

106 com/sun/media/sound/SoftProvider.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 javax.sound.midi.MidiDevice;
28 import javax.sound.midi.MidiDevice.Info;
29 import javax.sound.midi.spi.MidiDeviceProvider;
30
31 /**
32  * Software synthesizer provider class.
33  *
34  * @author Karl Helgason
35  */
36 public class SoftProvider extends MidiDeviceProvider {
37
38     protected final static Info softinfo = SoftSynthesizer.info;
39     private static Info[] softinfos = {softinfo};
40
41     public MidiDevice.Info[] getDeviceInfo() {
42         return softinfos;
43     }
44
45     public MidiDevice getDevice(MidiDevice.Info info) {
46         if (info == softinfo) {
47             return new SoftSynthesizer();
48         }
49         return null;
50     }
51 }
```

107 com/sun/media/sound/SoftReceiver.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.TreeMap;
28
29 import javax.sound.midi.MidiDevice;
30 import javax.sound.midi.MidiMessage;
31 import javax.sound.midi.ShortMessage;
32
33 /**
34 * Software synthesizer MIDI receiver class.
35 *
36 * @author Karl Helgason
37 */
38 public class SoftReceiver implements MidiDeviceReceiver {
39
40     protected boolean open = true;
41     private Object control_mutex;
42     private SoftSynthesizer synth;
43     protected TreeMap<Long, Object> midimessages;
44     protected SoftMainMixer mainmixer;
45
46     public SoftReceiver(SoftSynthesizer synth) {
47         this.control_mutex = synth.control_mutex;
48         this.synth = synth;
49         this.mainmixer = synth.getMainMixer();
50         if (mainmixer != null)
51             this.midimessages = mainmixer.midimessages;
52     }
53
54     public MidiDevice getMidiDevice() {
55         return synth;
56     }
57
58     public void send(MidiMessage message, long timeStamp) {
59
60         synchronized (control_mutex) {
```

```

61         if (!open)
62             throw new IllegalStateException("Receiver_is_not_open");
63     }
64
65     if (timeStamp != -1) {
66         synchronized (control_mutex) {
67             mainmixer.activity();
68             while (midimessages.get(timeStamp) != null)
69                 timeStamp++;
70             if (message instanceof ShortMessage
71                 && (((ShortMessage)message).getChannel() > 0xF)) {
72                 midimessages.put(timeStamp, message.clone());
73             } else {
74                 midimessages.put(timeStamp, message.getMessage());
75             }
76         }
77     } else {
78         mainmixer.processMessage(message);
79     }
80 }
81
82 public void close() {
83     synchronized (control_mutex) {
84         open = false;
85     }
86     synth.removeReceiver(this);
87 }
88 }

```

108 com/sun/media/sound/SoftResampler.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 /**
28 * Basic resampler interface.
29 *
30 * @author Karl Helgason
31 */
32 public interface SoftResampler {
33
34     public SoftResamplerStreamer openStreamer();
35 }
```


109 com/sun/media/sound/SoftResamplerStreamer.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.io.IOException;
28
29 /**
30  * Resampler stream interface.
31  *
32  * @author Karl Helgason
33  */
34 public interface SoftResamplerStreamer extends ModelOscillatorStream {
35
36     public void open(ModelWavetable osc, float outputsamplerate)
37         throws IOException;
38 }
```

110 com/sun/media/sound/SoftReverb.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.Arrays;
28
29 /**
30 * Reverb effect based on allpass/comb filters. First audio is send to 8
31 * parelled comb filters and then mixed together and then finally send thru 3
32 * different allpass filters.
33 *
34 * @author Karl Helgason
35 */
36 public class SoftReverb implements SoftAudioProcessor {
37
38     private final static class Delay {
39
40         private float[] delaybuffer;
41         private int rovepos = 0;
42
43         public Delay() {
44             delaybuffer = null;
45         }
46
47         public void setDelay(int delay) {
48             if (delay == 0)
49                 delaybuffer = null;
50             else
51                 delaybuffer = new float[delay];
52             rovepos = 0;
53         }
54
55         public void processReplace(float[] inout) {
56             if (delaybuffer == null)
57                 return;
58             int len = inout.length;
59             int rnlén = delaybuffer.length;
60             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;

```

```

185     }
186 }
187 private float roomsize;
188 private float damp;
189 private float gain = 1;
190 private Delay delay;
191 private Comb[] combL;
192 private Comb[] combR;
193 private AllPass[] allpassL;
194 private AllPass[] allpassR;
195 private float[] input;
196 private float[] out;
197 private float[] pre1;
198 private float[] pre2;
199 private float[] pre3;
200 private boolean denormal_flip = false;
201 private boolean mix = true;
202 private SoftAudioBuffer inputA;
203 private SoftAudioBuffer left;
204 private SoftAudioBuffer right;
205 private boolean dirty = true;
206 private float dirty_roomsize;
207 private float dirty_damp;
208 private float dirty_predelay;
209 private float dirty_gain;
210 private float samplerate;
211 private boolean light = true;
212
213 public void init(float samplerate, float controlrate) {
214     this.samplerate = samplerate;
215
216     double freqscale = ((double) samplerate) / 44100.0;
217     // freqscale = 1.0/ freqscale;
218
219     int stereospread = 23;
220
221     delay = new Delay();
222
223     combL = new Comb[8];
224     combR = new Comb[8];
225     combL[0] = new Comb((int) (freqscale * (1116)));
226     combR[0] = new Comb((int) (freqscale * (1116 + stereospread)));
227     combL[1] = new Comb((int) (freqscale * (1188)));
228     combR[1] = new Comb((int) (freqscale * (1188 + stereospread)));
229     combL[2] = new Comb((int) (freqscale * (1277)));
230     combR[2] = new Comb((int) (freqscale * (1277 + stereospread)));
231     combL[3] = new Comb((int) (freqscale * (1356)));
232     combR[3] = new Comb((int) (freqscale * (1356 + stereospread)));
233     combL[4] = new Comb((int) (freqscale * (1422)));
234     combR[4] = new Comb((int) (freqscale * (1422 + stereospread)));
235     combL[5] = new Comb((int) (freqscale * (1491)));
236     combR[5] = new Comb((int) (freqscale * (1491 + stereospread)));
237     combL[6] = new Comb((int) (freqscale * (1557)));
238     combR[6] = new Comb((int) (freqscale * (1557 + stereospread)));
239     combL[7] = new Comb((int) (freqscale * (1617)));
240     combR[7] = new Comb((int) (freqscale * (1617 + stereospread)));
241
242     allpassL = new AllPass[4];
243     allpassR = new AllPass[4];
244     allpassL[0] = new AllPass((int) (freqscale * (556)));
245     allpassR[0] = new AllPass((int) (freqscale * (556 + stereospread)));
246     allpassL[1] = new AllPass((int) (freqscale * (441)));

```

```

247 allpassR[1] = new AllPass((int) (freqscale * (441 + stereospread)));
248 allpassL[2] = new AllPass((int) (freqscale * (341)));
249 allpassR[2] = new AllPass((int) (freqscale * (341 + stereospread)));
250 allpassL[3] = new AllPass((int) (freqscale * (225)));
251 allpassR[3] = new AllPass((int) (freqscale * (225 + stereospread)));
252
253 for (int i = 0; i < allpassL.length; i++) {
254     allpassL[i].setFeedBack(0.5f);
255     allpassR[i].setFeedBack(0.5f);
256 }
257
258 /* Init other settings */
259 globalParameterControlChange(new int[]{0x01 * 128 + 0x01}, 0, 4);
260
261 }
262
263 public void setInput(int pin, SoftAudioBuffer input) {
264     if (pin == 0)
265         inputA = input;
266 }
267
268 public void setOutput(int pin, SoftAudioBuffer output) {
269     if (pin == 0)
270         left = output;
271     if (pin == 1)
272         right = output;
273 }
274
275 public void setMixMode(boolean mix) {
276     this.mix = mix;
277 }
278
279 private boolean silent = true;
280
281 public void processAudio() {
282     boolean silent_input = this.inputA.isSilent();
283     if(!silent_input)
284         silent = false;
285     if(silent)
286     {
287         if (!mix) {
288             left.clear();
289             right.clear();
290         }
291         return;
292     }
293
294     float[] inputA = this.inputA.array();
295     float[] left = this.left.array();
296     float[] right = this.right == null ? null : this.right.array();
297
298     int numsamples = inputA.length;
299     if (input == null || input.length < numsamples)
300         input = new float[numsamples];
301
302     float again = gain * 0.018f / 2;
303
304     denormal_flip = !denormal_flip;
305     if(denormal_flip)
306         for (int i = 0; i < numsamples; i++)
307             input[i] = inputA[i] * again + 1E-20f;
308     else

```

```

309     for (int i = 0; i < numsamples; i++)
310         input[i] = inputA[i] * again - 1E-20f;
311
312     delay.processReplace(input);
313
314     if(light && (right != null))
315     {
316         if (pre1 == null || pre1.length < numsamples)
317         {
318             pre1 = new float[numsamples];
319             pre2 = new float[numsamples];
320             pre3 = new float[numsamples];
321         }
322
323         for (int i = 0; i < allpassL.length; i++)
324             allpassL[i].processReplace(input);
325
326         combL[0].processReplace(input, pre3);
327         combL[1].processReplace(input, pre3);
328
329         combL[2].processReplace(input, pre1);
330         for (int i = 4; i < combL.length-2; i+=2)
331             combL[i].processMix(input, pre1);
332
333         combL[3].processReplace(input, pre2);
334         for (int i = 5; i < combL.length-2; i+=2)
335             combL[i].processMix(input, pre2);
336
337         if (!mix)
338         {
339             Arrays.fill(right, 0);
340             Arrays.fill(left, 0);
341         }
342         for (int i = combR.length-2; i < combR.length; i++)
343             combR[i].processMix(input, right);
344         for (int i = combL.length-2; i < combL.length; i++)
345             combL[i].processMix(input, left);
346
347         for (int i = 0; i < numsamples; i++)
348         {
349             float p = pre1[i] - pre2[i];
350             float m = pre3[i];
351             left[i] += m + p;
352             right[i] += m - p;
353         }
354     }
355     else
356     {
357         if (out == null || out.length < numsamples)
358             out = new float[numsamples];
359
360         if (right != null) {
361             if (!mix)
362                 Arrays.fill(right, 0);
363             allpassR[0].processReplace(input, out);
364             for (int i = 1; i < allpassR.length; i++)
365                 allpassR[i].processReplace(out);
366             for (int i = 0; i < combR.length; i++)
367                 combR[i].processMix(out, right);
368         }
369
370         if (!mix)

```

```

371     Arrays.fill(left, 0);
372     allpassL[0].processReplace(input, out);
373     for (int i = 1; i < allpassL.length; i++)
374         allpassL[i].processReplace(out);
375     for (int i = 0; i < combL.length; i++)
376         combL[i].processMix(out, left);
377 }

```

```

384     if (silent_input) {
385         silent = true;
386         for (int i = 0; i < numsamples; i++)
387         {
388             float v = left[i];
389             if(v > 1E-10 || v < -1E-10)
390             {
391                 silent = false;
392                 break;
393             }
394         }
395     }

```

```

396 }

```

```

398
399 public void globalParameterControlChange(int[] slothpath, long param,
400     long value) {
401     if (slothpath.length == 1) {
402         if (slothpath[0] == 0x01 * 128 + 0x01) {
403
404             if (param == 0) {
405                 if (value == 0) {
406                     // Small Room A small size room with a length
407                     // of 5m or so.
408                     dirty_roomsize = (1.1f);
409                     dirty_damp = (5000);
410                     dirty_predelay = (0);
411                     dirty_gain = (4);
412                     dirty = true;
413                 }
414                 if (value == 1) {
415                     // Medium Room A medium size room with a length
416                     // of 10m or so.
417                     dirty_roomsize = (1.3f);
418                     dirty_damp = (5000);
419                     dirty_predelay = (0);
420                     dirty_gain = (3);
421                     dirty = true;
422                 }
423                 if (value == 2) {
424                     // Large Room A large size room suitable for
425                     // live performances.
426                     dirty_roomsize = (1.5f);
427                     dirty_damp = (5000);
428                     dirty_predelay = (0);
429                     dirty_gain = (2);
430                     dirty = true;
431                 }
432                 if (value == 3) {

```



```

433         // Medium Hall A medium size concert hall.
434         dirty_roomsize = (1.8f);
435         dirty_damp = (24000);
436         dirty_predelay = (0.02f);
437         dirty_gain = (1.5f);
438         dirty = true;
439     }
440     if (value == 4) {
441         // Large Hall A large size concert hall
442         // suitable for a full orchestra.
443         dirty_roomsize = (1.8f);
444         dirty_damp = (24000);
445         dirty_predelay = (0.03f);
446         dirty_gain = (1.5f);
447         dirty = true;
448     }
449     if (value == 8) {
450         // Plate A plate reverb simulation.
451         dirty_roomsize = (1.3f);
452         dirty_damp = (2500);
453         dirty_predelay = (0);
454         dirty_gain = (6);
455         dirty = true;
456     }
457 } else if (param == 1) {
458     dirty_roomsize = ((float) (Math.exp((value - 40) * 0.025)));
459     dirty = true;
460 }
461
462 }
463
464 }
465
466 public void processControlLogic() {
467     if (dirty) {
468         dirty = false;
469         setRoomSize(dirty_roomsize);
470         setDamp(dirty_damp);
471         setPreDelay(dirty_predelay);
472         setGain(dirty_gain);
473     }
474 }
475
476 public void setRoomSize(float value) {
477     roomsize = 1 - (0.17f / value);
478
479     for (int i = 0; i < combL.length; i++) {
480         combL[i].feedback = roomsize;
481         combR[i].feedback = roomsize;
482     }
483 }
484
485 public void setPreDelay(float value) {
486     delay.setDelay((int)(value * samplerate));
487 }
488
489 public void setGain(float gain) {
490     this.gain = gain;
491 }
492
493 public void setDamp(float value) {
494     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

```
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 javax.sound.midi.InvalidMidiDataException;
28 import javax.sound.midi.ShortMessage;
29
30 /**
31 * A short message class that support for than 16 midi channels.
32 *
33 * @author Karl Helgason
34 */
35 public class SoftShortMessage extends ShortMessage {
36
37     int channel = 0;
38
39     public int getChannel() {
40         return channel;
41     }
42
43     public void setMessage(int command, int channel, int data1, int data2)
44         throws InvalidMidiDataException {
45         this.channel = channel;
46         super.setMessage(command, channel & 0xF, data1, data2);
47     }
48
49     public Object clone() {
50         SoftShortMessage clone = new SoftShortMessage();
51         try {
52             clone.setMessage(getCommand(), getChannel(), getData1(), getData2());
53         } catch (InvalidMidiDataException e) {
54             throw new IllegalArgumentException(e);
55         }
56         return clone;
57     }
58 }
```

112 com/sun/media/sound/SoftSincResampler.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 /**
28 * Hann windowed sinc interpolation resampler with anti-alias filtering.
29 *
30 * Using 30 points for the interpolation.
31 *
32 * @author Karl Helgason
33 */
34 public class SoftSincResampler extends SoftAbstractResampler {
35
36     float[][][] sinc_table;
37     int sinc_scale_size = 100;
38     int sinc_table_fsize = 800;
39     int sinc_table_size = 30;
40     int sinc_table_center = sinc_table_size / 2;
41
42     public SoftSincResampler() {
43         super();
44         sinc_table = new float[sinc_scale_size][sinc_table_fsize][];
45         for (int s = 0; s < sinc_scale_size; s++) {
46             float scale = (float) (1.0 / (1.0 + Math.pow(s, 1.1) / 10.0));
47             for (int i = 0; i < sinc_table_fsize; i++) {
48                 sinc_table[s][i] = sincTable(sinc_table_size,
49                     -i / ((float)sinc_table_fsize), scale);
50             }
51         }
52     }
53
54     // Normalized sinc function
55     public static double sinc(double x) {
56         return (x == 0.0) ? 1.0 : Math.sin(Math.PI * x) / (Math.PI * x);
57     }
58
59     // Generate hann window suitable for windowing sinc
60     public static float[] wHanning(int size, float offset) {
```

```

61     float[] window_table = new float[size];
62     for (int k = 0; k < size; k++) {
63         window_table[k] = (float)(-0.5
64             * Math.cos(2.0 * Math.PI * (double)(k + offset)
65                 / (double) size) + 0.5);
66     }
67     return window_table;
68 }
69
70 // Generate sinc table
71 public static float[] sincTable(int size, float offset, float scale) {
72     int center = size / 2;
73     float[] w = wHanning(size, offset);
74     for (int k = 0; k < size; k++)
75         w[k] *= sinc((-center + k + offset) * scale) * scale;
76     return w;
77 }
78
79 public int getPadding() // must be at least half of sinc_table_size
80 {
81     return sinc_table_size / 2 + 2;
82 }
83
84 public void interpolate(float[] in, float[] in_offset, float in_end,
85     float[] startpitch, float pitchstep, float[] out, int[] out_offset,
86     int out_end) {
87     float pitch = startpitch[0];
88     float ix = in_offset[0];
89     int ox = out_offset[0];
90     float ix_end = in_end;
91     int ox_end = out_end;
92     int max_p = sinc_scale_size - 1;
93     if (pitchstep == 0) {
94
95         int p = (int) ((pitch - 1) * 10.0f);
96         if (p < 0)
97             p = 0;
98         else if (p > max_p)
99             p = max_p;
100         float[][] sinc_table_f = this.sinc_table[p];
101         while (ix < ix_end && ox < ox_end) {
102             int iix = (int) ix;
103             float[] sinc_table =
104                 sinc_table_f[(int)((ix - iix) * sinc_table_fsize)];
105             int xx = iix - sinc_table_center;
106             float y = 0;
107             for (int i = 0; i < sinc_table_size; i++, xx++)
108                 y += in[xx] * sinc_table[i];
109             out[ox++] = y;
110             ix += pitch;
111         }
112     } else {
113         while (ix < ix_end && ox < ox_end) {
114             int iix = (int) ix;
115             int p = (int) ((pitch - 1) * 10.0f);
116             if (p < 0)
117                 p = 0;
118             else if (p > max_p)
119                 p = max_p;
120             float[][] sinc_table_f = this.sinc_table[p];
121
122             float[] sinc_table =

```

```

123         sinc_table_f[(int)((ix - iix) * sinc_table_fsize)];
124     int xx = iix - sinc_table_center;
125     float y = 0;
126     for (int i = 0; i < sinc_table_size; i++, xx++)
127         y += in[xx] * sinc_table[i];
128     out[ox++] = y;
129
130     ix += pitch;
131     pitch += pitchstep;
132 }
133 }
134 in_offset[0] = ix;
135 out_offset[0] = ox;
136 startpitch[0] = pitch;
137
138 }
139 }

```

113 com/sun/media/sound/SoftSynthesizer.java

```
1  /*
2   * Copyright 2008-2010 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
26 package com.sun.media.sound;
27
28 import java.io.BufferedInputStream;
29 import java.io.File;
30 import java.io.FileInputStream;
31 import java.io.FileOutputStream;
32 import java.io.IOException;
33 import java.io.InputStream;
34 import java.io.OutputStream;
35 import java.lang.ref.WeakReference;
36 import java.security.AccessController;
37 import java.security.PrivilegedAction;
38 import java.util.ArrayList;
39 import java.util.Arrays;
40 import java.util.HashMap;
41 import java.util.List;
42 import java.util.Map;
43 import java.util.Properties;
44 import java.util.StringTokenizer;
45 import java.util.prefs.BackingStoreException;
46 import java.util.prefs.Preferences;
47
48 import javax.sound.midi.Instrument;
49 import javax.sound.midi.MidiChannel;
50 import javax.sound.midi.MidiDevice;
51 import javax.sound.midi.MidiSystem;
52 import javax.sound.midi.MidiUnavailableException;
53 import javax.sound.midi.Patch;
54 import javax.sound.midi.Receiver;
55 import javax.sound.midi.Soundbank;
56 import javax.sound.midi.Transmitter;
57 import javax.sound.midi.VoiceStatus;
58 import javax.sound.sampled.AudioFormat;
59 import javax.sound.sampled.AudioInputStream;
60 import javax.sound.sampled.AudioSystem;
```

```

61 import javax.sound.sampled.LineUnavailableException;
62 import javax.sound.sampled.SourceDataLine;
63
64 /**
65  * The software synthesizer class.
66  *
67  * @author Karl Helgason
68  */
69 public class SoftSynthesizer implements AudioSynthesizer,
70     ReferenceCountingDevice {
71
72     protected static class WeakAudioStream extends InputStream
73     {
74         private volatile AudioInputStream stream;
75         public SoftAudioPusher pusher = null;
76         public AudioInputStream jitter_stream = null;
77         public SourceDataLine sourceDataLine = null;
78         public volatile long silent_samples = 0;
79         private int framesize = 0;
80         private WeakReference<AudioInputStream> weak_stream_link;
81         private AudioFloatConverter converter;
82         private float[] silentbuffer = null;
83         private int samplesize;
84
85         public void setInputStream(AudioInputStream stream)
86         {
87             this.stream = stream;
88         }
89
90         public int available() throws IOException {
91             AudioInputStream local_stream = stream;
92             if(local_stream != null)
93                 return local_stream.available();
94             return 0;
95         }
96
97         public int read() throws IOException {
98             byte[] b = new byte[1];
99             if (read(b) == -1)
100                 return -1;
101             return b[0] & 0xFF;
102         }
103
104         public int read(byte[] b, int off, int len) throws IOException {
105             AudioInputStream local_stream = stream;
106             if(local_stream != null)
107                 return local_stream.read(b, off, len);
108             else
109             {
110                 int flen = len / samplesize;
111                 if(silentbuffer == null || silentbuffer.length < flen)
112                     silentbuffer = new float[flen];
113                 converter.toByteArray(silentbuffer, flen, b, off);
114
115                 silent_samples += (long)((len / framesize));
116
117                 if(pusher != null)
118                     if(weak_stream_link.get() == null)
119                     {
120                         Runnable runnable = new Runnable()
121                         {
122                             SoftAudioPusher _pusher = pusher;

```



```

123         AudioInputStream _jitter_stream = jitter_stream;
124         SourceDataLine _sourceDataLine = sourceDataLine;
125         public void run()
126         {
127             _pusher.stop();
128             if(_jitter_stream != null)
129                 try {
130                     _jitter_stream.close();
131                 } catch (IOException e) {
132                     e.printStackTrace();
133                 }
134             if(_sourceDataLine != null)
135                 _sourceDataLine.close();
136         }
137     };
138     pusher = null;
139     jitter_stream = null;
140     sourceDataLine = null;
141     new Thread(runnable).start();
142 }
143 return len;
144 }
145 }
146
147 public WeakAudioStream(AudioInputStream stream) {
148     this.stream = stream;
149     weak_stream_link = new WeakReference<AudioInputStream>(stream);
150     converter = AudioFloatConverter.getConverter(stream.getFormat());
151     samplesize = stream.getFormat().getFrameSize() / stream.getFormat().getChannels();
152     framesize = stream.getFormat().getFrameSize();
153 }
154
155 public AudioInputStream getAudioInputStream()
156 {
157     return new AudioInputStream(this, stream.getFormat(), AudioSystem.NOT_SPECIFIED);
158 }
159
160 public void close() throws IOException
161 {
162     AudioInputStream astream = weak_stream_link.get();
163     if(astream != null)
164         astream.close();
165 }
166 }
167
168 private static class Info extends MidiDevice.Info {
169     public Info() {
170         super(INFO_NAME, INFO_VENDOR, INFO_DESCRIPTION, INFO_VERSION);
171     }
172 }
173
174 protected static final String INFO_NAME = "Gervill";
175 protected static final String INFO_VENDOR = "OpenJDK";
176 protected static final String INFO_DESCRIPTION = "Software_MIDI_Synthesizer";
177 protected static final String INFO_VERSION = "1.0";
178 protected final static MidiDevice.Info info = new Info();
179
180 private static SourceDataLine testline = null;
181
182 private static Soundbank defaultSoundBank = null;
183
184 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()) {

```

```

247         if (performer.getOscillators() != null) {
248             for (ModelOscillator osc : performer.getOscillators()) {
249                 if (osc instanceof ModelByteBufferWavetable) {
250                     ModelByteBufferWavetable w = (ModelByteBufferWavetable)osc;
251                     ModelByteBuffer buff = w.getBuffer();
252                     if (buff != null)
253                         buffers.add(buff);
254                     buff = w.get8BitExtensionBuffer();
255                     if (buff != null)
256                         buffers.add(buff);
257                 }
258             }
259         }
260     }
261 }
262
263 private boolean loadSamples(List<ModelInstrument> instruments) {
264     if (largemode)
265         return true;
266     List<ModelByteBuffer> buffers = new ArrayList<ModelByteBuffer>();
267     for (ModelInstrument instrument : instruments)
268         getBuffers(instrument, buffers);
269     try {
270         ModelByteBuffer.loadAll(buffers);
271     } catch (IOException e) {
272         return false;
273     }
274     return true;
275 }
276
277 private boolean loadInstruments(List<ModelInstrument> instruments) {
278     if (!isOpen())
279         return false;
280     if (!loadSamples(instruments))
281         return false;
282
283     synchronized (control_mutex) {
284         if (channels != null)
285             for (SoftChannel c : channels)
286                 {
287                     c.current_instrument = null;
288                     c.current_director = null;
289                 }
290         for (Instrument instrument : instruments) {
291             String pat = patchToString(instrument.getPatch());
292             SoftInstrument softins
293                 = new SoftInstrument((ModelInstrument) instrument);
294             inslist.put(pat, softins);
295             loadedlist.put(pat, (ModelInstrument) instrument);
296         }
297     }
298
299     return true;
300 }
301
302 private void processPropertyInfo(Map<String, Object> info) {
303     AudioSynthesizerPropertyInfo[] items = getPropertyInfo(info);
304
305     String resamplerType = (String)items[0].value;
306     if (resamplerType.equalsIgnoreCase("point"))
307     {
308         this.resampler = new SoftPointResampler();

```

```

309         this.resamplerType = "point";
310     }
311     else if (resamplerType.equalsIgnoreCase("linear"))
312     {
313         this.resampler = new SoftLinearResampler2();
314         this.resamplerType = "linear";
315     }
316     else if (resamplerType.equalsIgnoreCase("linear1"))
317     {
318         this.resampler = new SoftLinearResampler();
319         this.resamplerType = "linear1";
320     }
321     else if (resamplerType.equalsIgnoreCase("linear2"))
322     {
323         this.resampler = new SoftLinearResampler2();
324         this.resamplerType = "linear2";
325     }
326     else if (resamplerType.equalsIgnoreCase("cubic"))
327     {
328         this.resampler = new SoftCubicResampler();
329         this.resamplerType = "cubic";
330     }
331     else if (resamplerType.equalsIgnoreCase("lanczos"))
332     {
333         this.resampler = new SoftLanczosResampler();
334         this.resamplerType = "lanczos";
335     }
336     else if (resamplerType.equalsIgnoreCase("sinc"))
337     {
338         this.resampler = new SoftSincResampler();
339         this.resamplerType = "sinc";
340     }
341
342     setFormat((AudioFormat)items[2].value);
343     controlrate = (Float)items[1].value;
344     latency = (Long)items[3].value;
345     deviceid = (Integer)items[4].value;
346     maxpoly = (Integer)items[5].value;
347     reverb_on = (Boolean)items[6].value;
348     chorus_on = (Boolean)items[7].value;
349     agc_on = (Boolean)items[8].value;
350     largemode = (Boolean)items[9].value;
351     number_of_midi_channels = (Integer)items[10].value;
352     jitter_correction = (Boolean)items[11].value;
353     reverb_light = (Boolean)items[12].value;
354     load_default_soundbank = (Boolean)items[13].value;
355 }
356
357 private String patchToString(Patch patch) {
358     if (patch instanceof ModelPatch && ((ModelPatch) patch).isPercussion())
359         return "p." + patch.getProgram() + "." + patch.getBank();
360     else
361         return patch.getProgram() + "." + patch.getBank();
362 }
363
364 private void setFormat(AudioFormat format) {
365     if (format.getChannels() > 2) {
366         throw new IllegalArgumentException(
367             "Only mono and stereo audio supported.");
368     }
369     if (AudioFloatConverter.getConverter(format) == null)
370         throw new IllegalArgumentException("Audio format not supported.");

```

```

371     this.format = format;
372 }
373
374 protected void removeReceiver(Receiver recv) {
375     boolean perform_close = false;
376     synchronized (control_mutex) {
377         if (recvslist.remove(recv)) {
378             if (implicitOpen && recvslist.isEmpty())
379                 perform_close = true;
380         }
381     }
382     if (perform_close)
383         close();
384 }
385
386 protected SoftMainMixer getMainMixer() {
387     if (!isOpen())
388         return null;
389     return mainmixer;
390 }
391
392 protected SoftInstrument findInstrument(int program, int bank, int channel) {
393
394     // Add support for GM2 banks 0x78 and 0x79
395     // as specified in DLS 2.2 in Section 1.4.6
396     // which allows using percussion and melodic instruments
397     // on all channels
398     if (bank >> 7 == 0x78 || bank >> 7 == 0x79) {
399         SoftInstrument current_instrument
400             = inslist.get(program + "." + bank);
401         if (current_instrument != null)
402             return current_instrument;
403
404         String p_plaf;
405         if (bank >> 7 == 0x78)
406             p_plaf = "p.";
407         else
408             p_plaf = "";
409
410         // Instrument not found fallback to MSB:bank, LSB:0
411         current_instrument = inslist.get(p_plaf + program + "."
412             + ((bank & 128) << 7));
413         if (current_instrument != null)
414             return current_instrument;
415         // Instrument not found fallback to MSB:0, LSB:bank
416         current_instrument = inslist.get(p_plaf + program + "."
417             + (bank & 128));
418         if (current_instrument != null)
419             return current_instrument;
420         // Instrument not found fallback to MSB:0, LSB:0
421         current_instrument = inslist.get(p_plaf + program + ".0");
422         if (current_instrument != null)
423             return current_instrument;
424         // Instrument not found fallback to MSB:0, LSB:0, program=0
425         current_instrument = inslist.get(p_plaf + program + "0.0");
426         if (current_instrument != null)
427             return current_instrument;
428         return null;
429     }
430
431     // Channel 10 uses percussion instruments
432     String p_plaf;

```

```

433     if (channel == 9)
434         p_plaf = "p.";
435     else
436         p_plaf = "";
437
438     SoftInstrument current_instrument
439         = inslist.get(p_plaf + program + "." + bank);
440     if (current_instrument != null)
441         return current_instrument;
442     // Instrument not found fallback to MSB:0, LSB:0
443     current_instrument = inslist.get(p_plaf + program + ".0");
444     if (current_instrument != null)
445         return current_instrument;
446     // Instrument not found fallback to MSB:0, LSB:0, program=0
447     current_instrument = inslist.get(p_plaf + "0.0");
448     if (current_instrument != null)
449         return current_instrument;
450     return null;
451 }
452
453 protected int getVoiceAllocationMode() {
454     return voice_allocation_mode;
455 }
456
457 protected int getGeneralMidiMode() {
458     return gmmode;
459 }
460
461 protected void setGeneralMidiMode(int gmmode) {
462     this.gmmode = gmmode;
463 }
464
465 protected int getDeviceID() {
466     return deviceid;
467 }
468
469 protected float getControlRate() {
470     return controlrate;
471 }
472
473 protected SoftVoice[] getVoices() {
474     return voices;
475 }
476
477 protected SoftTuning getTuning(Patch patch) {
478     String t_id = patchToString(patch);
479     SoftTuning tuning = tunings.get(t_id);
480     if (tuning == null) {
481         tuning = new SoftTuning(patch);
482         tunings.put(t_id, tuning);
483     }
484     return tuning;
485 }
486
487 public long getLatency() {
488     synchronized (control_mutex) {
489         return latency;
490     }
491 }
492
493 public AudioFormat getFormat() {
494     synchronized (control_mutex) {

```

```

495         return format;
496     }
497 }
498
499 public int getMaxPolyphony() {
500     synchronized (control_mutex) {
501         return maxpoly;
502     }
503 }
504
505 public MidiChannel[] getChannels() {
506
507     synchronized (control_mutex) {
508         // if (external_channels == null) => the synthesizer is not open,
509         // create 16 proxy channels
510         // otherwise external_channels has the same length as channels array
511         if (external_channels == null) {
512             external_channels = new SoftChannelProxy[16];
513             for (int i = 0; i < external_channels.length; i++)
514                 external_channels[i] = new SoftChannelProxy();
515         }
516         MidiChannel[] ret;
517         if (isOpen())
518             ret = new MidiChannel[channels.length];
519         else
520             ret = new MidiChannel[16];
521         for (int i = 0; i < ret.length; i++)
522             ret[i] = external_channels[i];
523         return ret;
524     }
525 }
526
527 public VoiceStatus[] getVoiceStatus() {
528     if (!isOpen()) {
529         VoiceStatus[] tempVoiceStatusArray
530             = new VoiceStatus[getMaxPolyphony()];
531         for (int i = 0; i < tempVoiceStatusArray.length; i++) {
532             VoiceStatus b = new VoiceStatus();
533             b.active = false;
534             b.bank = 0;
535             b.channel = 0;
536             b.note = 0;
537             b.program = 0;
538             b.volume = 0;
539             tempVoiceStatusArray[i] = b;
540         }
541         return tempVoiceStatusArray;
542     }
543
544     synchronized (control_mutex) {
545         VoiceStatus[] tempVoiceStatusArray = new VoiceStatus[voices.length];
546         for (int i = 0; i < voices.length; i++) {
547             VoiceStatus a = voices[i];
548             VoiceStatus b = new VoiceStatus();
549             b.active = a.active;
550             b.bank = a.bank;
551             b.channel = a.channel;
552             b.note = a.note;
553             b.program = a.program;
554             b.volume = a.volume;
555             tempVoiceStatusArray[i] = b;
556         }

```

```

557         return tempVoiceStatusArray;
558     }
559 }
560
561 public boolean isSoundbankSupported(Soundbank soundbank) {
562     for (Instrument ins: soundbank.getInstruments())
563         if (!(ins instanceof ModelInstrument))
564             return false;
565     return true;
566 }
567
568 public boolean loadInstrument(Instrument instrument) {
569     if (instrument == null || !(instrument instanceof ModelInstrument)) {
570         throw new IllegalArgumentException("Unsupported instrument: " +
571             instrument);
572     }
573     List<ModelInstrument> instruments = new ArrayList<ModelInstrument>();
574     instruments.add((ModelInstrument)instrument);
575     return loadInstruments(instruments);
576 }
577
578 public void unloadInstrument(Instrument instrument) {
579     if (instrument == null || !(instrument instanceof ModelInstrument)) {
580         throw new IllegalArgumentException("Unsupported instrument: " +
581             instrument);
582     }
583     if (!isOpen())
584         return;
585
586     String pat = patchToString(instrument.getPatch());
587     synchronized (control_mutex) {
588         for (SoftChannel c: channels)
589             c.current_instrument = null;
590         inslist.remove(pat);
591         loadedlist.remove(pat);
592         for (int i = 0; i < channels.length; i++) {
593             channels[i].allSoundOff();
594         }
595     }
596 }
597
598 public boolean remapInstrument(Instrument from, Instrument to) {
599
600     if (from == null)
601         throw new NullPointerException();
602     if (to == null)
603         throw new NullPointerException();
604     if (!(from instanceof ModelInstrument)) {
605         throw new IllegalArgumentException("Unsupported instrument: " +
606             from.toString());
607     }
608     if (!(to instanceof ModelInstrument)) {
609         throw new IllegalArgumentException("Unsupported instrument: " +
610             to.toString());
611     }
612     if (!isOpen())
613         return false;
614
615     synchronized (control_mutex) {
616         if (!loadedlist.containsValue(to))
617             throw new IllegalArgumentException("Instrument to is not loaded.");
618         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                 }

```

```

681         return null;
682     }
683 });
684
685 actions.add(new PrivilegedAction<InputStream>() {
686     public InputStream run() {
687         /*
688          * Try to load saved generated soundbank
689          */
690         File userhome = new File(System.getProperty("user.home"),
691             ".gervill");
692         File emg_soundbank_file = new File(userhome,
693             "soundbank-emg.sf2");
694         if (emg_soundbank_file.exists()) {
695             try {
696                 return new FileInputStream(emg_soundbank_file);
697             } catch (IOException e) {
698             }
699         }
700         return null;
701     }
702 });
703
704 for (PrivilegedAction<InputStream> action : actions) {
705     try {
706         InputStream is = AccessController.doPrivileged(action);
707         if(is == null) continue;
708         Soundbank sbk;
709         try {
710             sbk = MidiSystem.getSoundbank(new BufferedInputStream(is));
711         } finally {
712             is.close();
713         }
714         if (sbk != null) {
715             defaultSoundBank = sbk;
716             return defaultSoundBank;
717         }
718     } catch (Exception e) {
719     }
720 }
721
722 try {
723     /*
724      * Generate emergency soundbank
725      */
726     defaultSoundBank = EmergencySoundbank.createSoundbank();
727 } catch (Exception e) {
728 }
729
730 if (defaultSoundBank != null) {
731     /*
732      * Save generated soundbank to disk for faster future use.
733      */
734     OutputStream out = AccessController
735         .doPrivileged(new PrivilegedAction<OutputStream>() {
736             public OutputStream run() {
737                 try {
738                     File userhome = new File(System
739                         .getProperty("user.home"),
740                         ".gervill");
741                     if (!userhome.exists())
742                         userhome.mkdirs();

```

```

743         File emg_soundbank_file = new File(
744             userhome, "soundbank-emg.sf2");
745         if (emg_soundbank_file.exists())
746             return null;
747         return new FileOutputStream(
748             emg_soundbank_file);
749     } catch (IOException e) {
750     } catch (SecurityException e) {
751     }
752     return null;
753 }
754 });
755 if (out != null) {
756     try {
757         ((SF2Soundbank) defaultSoundBank).save(out);
758         out.close();
759     } catch (IOException e) {
760     }
761 }
762 }
763 }
764 return defaultSoundBank;
765 }
766
767 public Instrument[] getAvailableInstruments() {
768     Soundbank defsbk = getDefaultSoundbank();
769     if (defsbk == null)
770         return new Instrument[0];
771     Instrument[] inslist_array = defsbk.getInstruments();
772     Arrays.sort(inslist_array, new ModelInstrumentComparator());
773     return inslist_array;
774 }
775
776 public Instrument[] getLoadedInstruments() {
777     if (!isOpen())
778         return new Instrument[0];
779
780     synchronized (control_mutex) {
781         ModelInstrument[] inslist_array =
782             new ModelInstrument[loadedlist.values().size()];
783         loadedlist.values().toArray(inslist_array);
784         Arrays.sort(inslist_array, new ModelInstrumentComparator());
785         return inslist_array;
786     }
787 }
788
789 public boolean loadAllInstruments(Soundbank soundbank) {
790     List<ModelInstrument> instruments = new ArrayList<ModelInstrument>();
791     for (Instrument ins: soundbank.getInstruments()) {
792         if (ins == null || !(ins instanceof ModelInstrument)) {
793             throw new IllegalArgumentException(
794                 "Unsupported_instrument:_" + ins);
795         }
796         instruments.add((ModelInstrument)ins);
797     }
798     return loadInstruments(instruments);
799 }
800
801 public void unloadAllInstruments(Soundbank soundbank) {
802     if (soundbank == null || !isSoundbankSupported(soundbank))
803         throw new IllegalArgumentException("Unsupported_soundbank:_" + soundbank);
804 }

```

```

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                 }
867             }
868         });
869 }

```

```

867         return p;
868     }
869     });
870 }
871
872 public AudioSynthesizerPropertyInfo[] getPropertyInfo(Map<String, Object> info) {
873     List<AudioSynthesizerPropertyInfo> list =
874         new ArrayList<AudioSynthesizerPropertyInfo>();
875
876     AudioSynthesizerPropertyInfo item;
877
878     // If info != null or synthesizer is closed
879     // we return how the synthesizer will be set on next open
880     // If info == null and synthesizer is open
881     // we return current synthesizer properties.
882     boolean o = info == null && open;
883
884     item = new AudioSynthesizerPropertyInfo("interpolation", o?resamplerType:"linear");
885     item.choices = new String[]{ "linear", "linear1", "linear2", "cubic",
886                                   "lanczos", "sinc", "point" };
887     item.description = "Interpolation_method";
888     list.add(item);
889
890     item = new AudioSynthesizerPropertyInfo("control_rate", o?controlrate:147f);
891     item.description = "Control_rate";
892     list.add(item);
893
894     item = new AudioSynthesizerPropertyInfo("format",
895         o?format:new AudioFormat(44100, 16, 2, true, false));
896     item.description = "Default_audio_format";
897     list.add(item);
898
899     item = new AudioSynthesizerPropertyInfo("latency", o?latency:120000L);
900     item.description = "Default_latency";
901     list.add(item);
902
903     item = new AudioSynthesizerPropertyInfo("device_id", o?deviceid:0);
904     item.description = "Device_ID_for_SysEx_Messages";
905     list.add(item);
906
907     item = new AudioSynthesizerPropertyInfo("max_polyphony", o?maxpoly:64);
908     item.description = "Maximum_polyphony";
909     list.add(item);
910
911     item = new AudioSynthesizerPropertyInfo("reverb", o?reverb_on:true);
912     item.description = "Turn_reverb_effect_on_or_off";
913     list.add(item);
914
915     item = new AudioSynthesizerPropertyInfo("chorus", o?chorus_on:true);
916     item.description = "Turn_chorus_effect_on_or_off";
917     list.add(item);
918
919     item = new AudioSynthesizerPropertyInfo("auto_gain_control", o?agc_on:true);
920     item.description = "Turn_auto_gain_control_on_or_off";
921     list.add(item);
922
923     item = new AudioSynthesizerPropertyInfo("large_mode", o?largemode:false);
924     item.description = "Turn_large_mode_on_or_off.";
925     list.add(item);
926
927     item = new AudioSynthesizerPropertyInfo("midi_channels", o?channels.length:16);
928     item.description = "Number_of_midi_channels.";

```

```

929 list.add(item);
930
931 item = new AudioSynthesizerPropertyInfo("jitter_correction", o?jitter_correction:true);
932 item.description = "Turn_jitter_correction_on_or_off.";
933 list.add(item);
934
935 item = new AudioSynthesizerPropertyInfo("light_reverb", o?reverb_light:true);
936 item.description = "Turn_light_reverb_mode_on_or_off";
937 list.add(item);
938
939 item = new AudioSynthesizerPropertyInfo("load_default_soundbank", o?
    load_default_soundbank:true);
940 item.description = "Enabled/disable_loading_default_soundbank";
941 list.add(item);
942
943 AudioSynthesizerPropertyInfo[] items;
944 items = list.toArray(new AudioSynthesizerPropertyInfo[list.size()]);
945
946 Properties storedProperties = getStoredProperties();
947
948 for (AudioSynthesizerPropertyInfo item2 : items) {
949     Object v = (info == null) ? null : info.get(item2.name);
950     v = (v != null) ? v : storedProperties.getProperty(item2.name);
951     if (v != null) {
952         Class c = (item2.valueClass);
953         if (c.isInstance(v))
954             item2.value = v;
955         else if (v instanceof String) {
956             String s = (String) v;
957             if (c == Boolean.class) {
958                 if (s.equalsIgnoreCase("true"))
959                     item2.value = Boolean.TRUE;
960                 if (s.equalsIgnoreCase("false"))
961                     item2.value = Boolean.FALSE;
962             } else if (c == AudioFormat.class) {
963                 int channels = 2;
964                 boolean signed = true;
965                 boolean bigendian = false;
966                 int bits = 16;
967                 float sampleRate = 44100f;
968                 try {
969                     StringTokenizer st = new StringTokenizer(s, ",_");
970                     String prevToken = "";
971                     while (st.hasMoreTokens()) {
972                         String token = st.nextToken().toLowerCase();
973                         if (token.equals("mono"))
974                             channels = 1;
975                         if (token.startsWith("channel"))
976                             channels = Integer.parseInt(prevToken);
977                         if (token.contains("unsigned"))
978                             signed = false;
979                         if (token.equals("big-endian"))
980                             bigendian = true;
981                         if (token.equals("bit"))
982                             bits = Integer.parseInt(prevToken);
983                         if (token.equals("hz"))
984                             sampleRate = Float.parseFloat(prevToken);
985                         prevToken = token;
986                     }
987                     item2.value = new AudioFormat(sampleRate, bits,
988                         channels, signed, bigendian);
989                 } catch (NumberFormatException e) {

```

```

990     }
991
992     } else
993     try {
994         if (c == Byte.class)
995             item2.value = Byte.valueOf(s);
996         else if (c == Short.class)
997             item2.value = Short.valueOf(s);
998         else if (c == Integer.class)
999             item2.value = Integer.valueOf(s);
1000        else if (c == Long.class)
1001            item2.value = Long.valueOf(s);
1002        else if (c == Float.class)
1003            item2.value = Float.valueOf(s);
1004        else if (c == Double.class)
1005            item2.value = Double.valueOf(s);
1006    } catch (NumberFormatException e) {
1007    }
1008    } else if (v instanceof Number) {
1009        Number n = (Number) v;
1010        if (c == Byte.class)
1011            item2.value = Byte.valueOf(n.byteValue());
1012        if (c == Short.class)
1013            item2.value = Short.valueOf(n.shortValue());
1014        if (c == Integer.class)
1015            item2.value = Integer.valueOf(n.intValue());
1016        if (c == Long.class)
1017            item2.value = Long.valueOf(n.longValue());
1018        if (c == Float.class)
1019            item2.value = Float.valueOf(n.floatValue());
1020        if (c == Double.class)
1021            item2.value = Double.valueOf(n.doubleValue());
1022    }
1023    }
1024    }
1025
1026    return items;
1027 }
1028
1029 public void open() throws MidiUnavailableException {
1030     if (isOpen()) {
1031         synchronized (control_mutex) {
1032             implicitOpen = false;
1033         }
1034         return;
1035     }
1036     open(null, null);
1037 }
1038
1039 public void open(SourceDataLine line, Map<String, Object> info) throws
MidiUnavailableException {
1040     if (isOpen()) {
1041         synchronized (control_mutex) {
1042             implicitOpen = false;
1043         }
1044         return;
1045     }
1046     synchronized (control_mutex) {
1047         Throwable causeException = null;
1048         try {
1049             if (line != null) {
1050                 // can throw IllegalArgumentException

```

```

1051         setFormat(line.getFormat());
1052     }
1053
1054     AudioInputStream ais = openStream(getFormat(), info);
1055
1056     weakstream = new WeakAudioStream(ais);
1057     ais = weakstream.getAudioInputStream();
1058
1059     if (line == null)
1060     {
1061         if (testline != null) {
1062             line = testline;
1063         } else {
1064             // can throw LineUnavailableException,
1065             // IllegalArgumentException, SecurityException
1066             line = AudioSystem.getSourceDataLine(getFormat());
1067         }
1068     }
1069
1070     double latency = this.latency;
1071
1072     if (!line.isOpen()) {
1073         int bufferSize = getFormat().getFrameSize()
1074             * (int)(getFormat().getFrameRate() * (latency/1000000f));
1075         // can throw LineUnavailableException,
1076         // IllegalArgumentException, SecurityException
1077         line.open(getFormat(), bufferSize);
1078
1079         // Remember that we opened that line
1080         // so we can close again in SoftSynthesizer.close()
1081         sourceDataLine = line;
1082     }
1083     if (!line.isActive())
1084         line.start();
1085
1086     int controlbuffersize = 512;
1087     try {
1088         controlbuffersize = ais.available();
1089     } catch (IOException e) {
1090     }
1091
1092     // Tell mixer not fill read buffers fully.
1093     // This lowers latency, and tells DataPusher
1094     // to read in smaller amounts.
1095     //mainmixer.readfully= false;
1096     //pusher = new DataPusher(line, ais);
1097
1098     int buffersize = line.getBufferSize();
1099     buffersize -= buffersize % controlbuffersize;
1100
1101     if (buffersize < 3 * controlbuffersize)
1102         buffersize = 3 * controlbuffersize;
1103
1104     if (jitter_correction) {
1105         ais = new SoftJitterCorrector(ais, buffersize,
1106             controlbuffersize);
1107         if(weakstream != null)
1108             weakstream.jitter_stream = ais;
1109     }
1110     pusher = new SoftAudioPusher(line, ais, controlbuffersize);
1111     pusher_stream = ais;
1112     pusher.start();

```



```

1113         if(weakstream != null)
1114         {
1115             weakstream.pusher = pusher;
1116             weakstream.sourceDataLine = sourceDataLine;
1117         }
1118     }
1119
1120     } catch (LineUnavailableException e) {
1121         causeException = e;
1122     } catch (IllegalArgumentException e) {
1123         causeException = e;
1124     } catch (SecurityException e) {
1125         causeException = e;
1126     }
1127
1128     if (causeException != null) {
1129         if (isOpen())
1130             close();
1131         // am: need MidiUnavailableException(Throwable) ctor!
1132         MidiUnavailableException ex = new MidiUnavailableException(
1133             "Can_not_open_line");
1134         ex.initCause(causeException);
1135         throw ex;
1136     }
1137
1138     }
1139 }
1140
1141 public AudioInputStream openStream(AudioFormat targetFormat,
1142     Map<String, Object> info) throws MidiUnavailableException {
1143
1144     if (isOpen())
1145         throw new MidiUnavailableException("Synthesizer_is_already_open");
1146
1147     synchronized (control_mutex) {
1148
1149         gmmode = 0;
1150         voice_allocation_mode = 0;
1151
1152         processPropertyInfo(info);
1153
1154         open = true;
1155         implicitOpen = false;
1156
1157         if (targetFormat != null)
1158             setFormat(targetFormat);
1159
1160         if (load_default_soundbank)
1161         {
1162             Soundbank defbank = getDefaultSoundbank();
1163             if (defbank != null) {
1164                 loadAllInstruments(defbank);
1165             }
1166         }
1167
1168         voices = new SoftVoice[maxpoly];
1169         for (int i = 0; i < maxpoly; i++)
1170             voices[i] = new SoftVoice(this);
1171
1172         mainmixer = new SoftMainMixer(this);
1173
1174         channels = new SoftChannel[number_of_midi_channels];

```

```

1175     for (int i = 0; i < channels.length; i++)
1176         channels[i] = new SoftChannel(this, i);
1177
1178     if (external_channels == null) {
1179         // Always create external_channels array
1180         // with 16 or more channels
1181         // so getChannels works correctly
1182         // when the synhtesizer is closed.
1183         if (channels.length < 16)
1184             external_channels = new SoftChannelProxy[16];
1185         else
1186             external_channels = new SoftChannelProxy[channels.length];
1187         for (int i = 0; i < external_channels.length; i++)
1188             external_channels[i] = new SoftChannelProxy();
1189     } else {
1190         // We must resize external_channels array
1191         // but we must also copy the old SoftChannelProxy
1192         // into the new one
1193         if (channels.length > external_channels.length) {
1194             SoftChannelProxy[] new_external_channels
1195                 = new SoftChannelProxy[channels.length];
1196             for (int i = 0; i < external_channels.length; i++)
1197                 new_external_channels[i] = external_channels[i];
1198             for (int i = external_channels.length;
1199                  i < new_external_channels.length; i++) {
1200                 new_external_channels[i] = new SoftChannelProxy();
1201             }
1202         }
1203     }
1204
1205     for (int i = 0; i < channels.length; i++)
1206         external_channels[i].setChannel(channels[i]);
1207
1208     for (SoftVoice voice: getVoices())
1209         voice.resampler = resampler.openStreamer();
1210
1211     for (Receiver recv: getReceivers()) {
1212         SoftReceiver srecv = ((SoftReceiver)recv);
1213         srecv.open = open;
1214         srecv.mainmixer = mainmixer;
1215         srecv.midimessages = mainmixer.midimessages;
1216     }
1217
1218     return mainmixer.getInputStream();
1219 }
1220
1221
1222 public void close() {
1223
1224     if (!isOpen())
1225         return;
1226
1227     SoftAudioPusher pusher_to_be_closed = null;
1228     AudioInputStream pusher_stream_to_be_closed = null;
1229     synchronized (control_mutex) {
1230         if (pusher != null) {
1231             pusher_to_be_closed = pusher;
1232             pusher_stream_to_be_closed = pusher_stream;
1233             pusher = null;
1234             pusher_stream = null;
1235         }
1236     }

```

```

1237
1238     if (pusher_to_be_closed != null) {
1239         // Pusher must not be closed synchronized against control_mutex,
1240         // this may result in synchronized conflict between pusher
1241         // and current thread.
1242         pusher_to_be_closed.stop();
1243
1244         try {
1245             pusher_stream_to_be_closed.close();
1246         } catch (IOException e) {
1247             //e.printStackTrace();
1248         }
1249     }
1250
1251     synchronized (control_mutex) {
1252
1253         if (mainmixer != null)
1254             mainmixer.close();
1255         open = false;
1256         implicitOpen = false;
1257         mainmixer = null;
1258         voices = null;
1259         channels = null;
1260
1261         if (external_channels != null)
1262             for (int i = 0; i < external_channels.length; i++)
1263                 external_channels[i].setChannel(null);
1264
1265         if (sourceDataLine != null) {
1266             sourceDataLine.close();
1267             sourceDataLine = null;
1268         }
1269
1270         inslist.clear();
1271         loadedlist.clear();
1272         tunings.clear();
1273
1274         while (recvslist.size() != 0)
1275             recvslist.get(recvslist.size() - 1).close();
1276     }
1277 }
1278
1279
1280 public boolean isOpen() {
1281     synchronized (control_mutex) {
1282         return open;
1283     }
1284 }
1285
1286 public long getMicrosecondPosition() {
1287
1288     if (!isOpen())
1289         return 0;
1290
1291     synchronized (control_mutex) {
1292         return mainmixer.getMicrosecondPosition();
1293     }
1294 }
1295
1296 public int getMaxReceivers() {
1297     return -1;
1298 }

```

```

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

```
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.io.UnsupportedEncodingException;
28
29 import javax.sound.midi.Patch;
30
31 /**
32  * A tuning program container, for use with MIDI Tuning.
33  * See: http://www.midi.org
34  *
35  * @author Karl Helgason
36  */
37 public class SoftTuning {
38
39     private String name = null;
40     private double[] tuning = new double[128];
41     private Patch patch = null;
42
43     public SoftTuning() {
44         name = "12-TET";
45         for (int i = 0; i < tuning.length; i++)
46             tuning[i] = i * 100;
47     }
48
49     public SoftTuning(byte[] data) {
50         for (int i = 0; i < tuning.length; i++)
51             tuning[i] = i * 100;
52         load(data);
53     }
54
55     public SoftTuning(Patch patch) {
56         this.patch = patch;
57         name = "12-TET";
58         for (int i = 0; i < tuning.length; i++)
59             tuning[i] = i * 100;
60     }
61 }
```

```

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;

```

```

123         int xx = data[r++] & 0xFF;
124         int yy = data[r++] & 0xFF;
125         int zz = data[r++] & 0xFF;
126         if (!(xx == 127 && yy == 127 && zz == 127))
127             tuning[kk] = 100.0*((xx*16384) + (yy*128) + zz)/16384.0);
128     }
129     break;
130 }
131 case 0x04: // KEY-BASED TUNING DUMP (NON-REAL-TIME)
132 {
133     // http://www.midi.org/about-midi/tuning_extens.shtml
134     if (!checksumOK(data))
135         break;
136     try {
137         name = new String(data, 7, 16, "ascii");
138     } catch (UnsupportedEncodingException e) {
139         name = null;
140     }
141     int r = 23;
142     for (int i = 0; i < 128; i++) {
143         int xx = data[r++] & 0xFF;
144         int yy = data[r++] & 0xFF;
145         int zz = data[r++] & 0xFF;
146         if (!(xx == 127 && yy == 127 && zz == 127))
147             tuning[i] = 100.0*((xx*16384) + (yy*128) + zz)/16384.0);
148     }
149     break;
150 }
151 case 0x05: // SCALE/OCTAVE TUNING DUMP, 1 byte format
152             // (NON-REAL-TIME)
153 {
154     // http://www.midi.org/about-midi/tuning_extens.shtml
155     if (!checksumOK(data))
156         break;
157     try {
158         name = new String(data, 7, 16, "ascii");
159     } catch (UnsupportedEncodingException e) {
160         name = null;
161     }
162     int[] octave_tuning = new int[12];
163     for (int i = 0; i < 12; i++)
164         octave_tuning[i] = (data[i + 23] & 0xFF) - 64;
165     for (int i = 0; i < tuning.length; i++)
166         tuning[i] = i * 100 + octave_tuning[i % 12];
167     break;
168 }
169 case 0x06: // SCALE/OCTAVE TUNING DUMP, 2 byte format
170             // (NON-REAL-TIME)
171 {
172     // http://www.midi.org/about-midi/tuning_extens.shtml
173     if (!checksumOK(data))
174         break;
175     try {
176         name = new String(data, 7, 16, "ascii");
177     } catch (UnsupportedEncodingException e) {
178         name = null;
179     }
180     double[] octave_tuning = new double[12];
181     for (int i = 0; i < 12; i++) {
182         int v = (data[i * 2 + 23] & 0xFF) * 128
183                 + (data[i * 2 + 24] & 0xFF);
184         octave_tuning[i] = (v / 8192.0 - 1) * 100.0;

```

```

185     }
186     for (int i = 0; i < tuning.length; i++)
187         tuning[i] = i * 100 + octave_tuning[i % 12];
188     break;
189 }
190 case 0x07: // SINGLE NOTE TUNING CHANGE (NON
191             // REAL-TIME/REAL-TIME) (BANK)
192             // http://www.midi.org/about-midi/tuning_extens.shtml
193     int ll = data[7] & 0xFF;
194     int r = 8;
195     for (int i = 0; i < ll; i++) {
196         int kk = data[r++] & 0xFF;
197         int xx = data[r++] & 0xFF;
198         int yy = data[r++] & 0xFF;
199         int zz = data[r++] & 0xFF;
200         if (!(xx == 127 && yy == 127 && zz == 127))
201             tuning[kk] = 100.0
202                 * (((xx*16384) + (yy*128) + zz) / 16384.0);
203     }
204     break;
205 case 0x08: // scale/octave tuning 1-byte form (Non
206             // Real-Time/REAL-TIME)
207     {
208         // http://www.midi.org/about-midi/tuning-scale.shtml
209         int[] octave_tuning = new int[12];
210         for (int i = 0; i < 12; i++)
211             octave_tuning[i] = (data[i + 8] & 0xFF) - 64;
212         for (int i = 0; i < tuning.length; i++)
213             tuning[i] = i * 100 + octave_tuning[i % 12];
214         break;
215     }
216 case 0x09: // scale/octave tuning 2-byte form (Non
217             // Real-Time/REAL-TIME)
218     {
219         // http://www.midi.org/about-midi/tuning-scale.shtml
220         double[] octave_tuning = new double[12];
221         for (int i = 0; i < 12; i++) {
222             int v = (data[i * 2 + 8] & 0xFF) * 128
223                 + (data[i * 2 + 9] & 0xFF);
224             octave_tuning[i] = (v / 8192.0 - 1) * 100.0;
225         }
226         for (int i = 0; i < tuning.length; i++)
227             tuning[i] = i * 100 + octave_tuning[i % 12];
228         break;
229     }
230 default:
231     break;
232 }
233 }
234 }
235 }
236
237 public double[] getTuning() {
238     return tuning;
239 }
240
241 public double getTuning(int noteNumber) {
242     return tuning[noteNumber];
243 }
244
245 public Patch getPatch() {
246     return patch;

```



```
247     }
248
249     public String getName() {
250         return name;
251     }
252
253     public void setName(String name) {
254         this.name = name;
255     }
256 }
```

115 com/sun/media/sound/SoftVoice.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.io.IOException;
28 import java.util.Arrays;
29 import java.util.HashMap;
30 import java.util.Map;
31
32 import javax.sound.midi.VoiceStatus;
33
34 /**
35  * Software synthesizer voice class.
36  *
37  * @author Karl Helgason
38  */
39 public class SoftVoice extends VoiceStatus {
40
41     public int exclusiveClass = 0;
42     public boolean releaseTriggered = false;
43     private int noteOn_noteNumber = 0;
44     private int noteOn_velocity = 0;
45     private int noteOff_velocity = 0;
46     private int delay = 0;
47     protected ModelChannelMixer channelmixer = null;
48     protected double tunedKey = 0;
49     protected SoftTuning tuning = null;
50     protected SoftChannel stealer_channel = null;
51     protected ModelConnectionBlock[] stealer_extendedConnectionBlocks = null;
52     protected SoftPerformer stealer_performer = null;
53     protected ModelChannelMixer stealer_channelmixer = null;
54     protected int stealer_voiceID = -1;
55     protected int stealer_noteNumber = 0;
56     protected int stealer_velocity = 0;
57     protected boolean stealer_releaseTriggered = false;
58     protected int voiceID = -1;
59     protected boolean sustain = false;
60     protected boolean sostenuto = false;
```

```

61     protected boolean portamento = false;
62     private SoftFilter filter_left;
63     private SoftFilter filter_right;
64     private SoftProcess eg = new SoftEnvelopeGenerator();
65     private SoftProcess lfo = new SoftLowFrequencyOscillator();
66     protected Map<String, SoftControl> objects =
67         new HashMap<String, SoftControl>();
68     protected SoftSynthesizer synthesizer;
69     protected SoftInstrument instrument;
70     protected SoftPerformer performer;
71     protected SoftChannel softchannel = null;
72     protected boolean on = false;
73     private boolean audiostarted = false;
74     private boolean started = false;
75     private boolean stopping = false;
76     private float osc_attenuation = 0.0f;
77     private ModelOscillatorStream osc_stream;
78     private int osc_stream_nrofchannels;
79     private float[][] osc_buff = new float[2][];
80     private boolean osc_stream_off_transmitted = false;
81     private boolean out_mixer_end = false;
82     private float out_mixer_left = 0;
83     private float out_mixer_right = 0;
84     private float out_mixer_effect1 = 0;
85     private float out_mixer_effect2 = 0;
86     private float last_out_mixer_left = 0;
87     private float last_out_mixer_right = 0;
88     private float last_out_mixer_effect1 = 0;
89     private float last_out_mixer_effect2 = 0;
90     protected ModelConnectionBlock[] extendedConnectionBlocks = null;
91     private ModelConnectionBlock[] connections;
92     // Last value added to destination
93     private double[] connections_last = new double[50];
94     // Pointer to source value
95     private double[][][] connections_src = new double[50][3][];
96     // Key-based override (if any)
97     private int[][] connections_src_kc = new int[50][3];
98     // Pointer to destination value
99     private double[][] connections_dst = new double[50][];
100    private boolean soundoff = false;
101    private float lastMuteValue = 0;
102    private float lastSoloMuteValue = 0;
103    protected double[] co_noteon_keynumber = new double[1];
104    protected double[] co_noteon_velocity = new double[1];
105    protected double[] co_noteon_on = new double[1];
106    private SoftControl co_noteon = new SoftControl() {
107        double[] keynumber = co_noteon_keynumber;
108        double[] velocity = co_noteon_velocity;
109        double[] on = co_noteon_on;
110        public double[] get(int instance, String name) {
111            if (name == null)
112                return null;
113            if (name.equals("keynumber"))
114                return keynumber;
115            if (name.equals("velocity"))
116                return velocity;
117            if (name.equals("on"))
118                return on;
119            return null;
120        }
121    };
122    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;

```

```

247     }
248
249     private void processConnection(int ix) {
250         ModelConnectionBlock conn = connections[ix];
251         double[][] src = connections_src[ix];
252         double[] dst = connections_dst[ix];
253         if (dst == null || Double.isInfinite(dst[0]))
254             return;
255
256         double value = conn.getScale();
257         if (softchannel.keybasedcontroller_active == null) {
258             ModelSource[] srcs = conn.getSources();
259             for (int i = 0; i < srcs.length; i++) {
260                 value *= transformValue(src[i][0], srcs[i]);
261                 if (value == 0)
262                     break;
263             }
264         } else {
265             ModelSource[] srcs = conn.getSources();
266             int[] src_kc = connections_src_kc[ix];
267             for (int i = 0; i < srcs.length; i++) {
268                 value *= transformValue(processKeyBasedController(src[i][0],
269                     src_kc[i]), srcs[i]);
270                 if (value == 0)
271                     break;
272             }
273         }
274
275         value = transformValue(value, conn.getDestination());
276         dst[0] = dst[0] - connections_last[ix] + value;
277         connections_last[ix] = value;
278         // co_mixer_gain[0] = 0;
279     }
280
281     protected void updateTuning(SoftTuning newtuning) {
282         tuning = newtuning;
283         tunedKey = tuning.getTuning(note) / 100.0;
284         if (!portamento) {
285             co_noteon_keynumber[0] = tunedKey * (1.0 / 128.0);
286             if (performer == null)
287                 return;
288             int[] c = performer.midi_connections[4];
289             if (c == null)
290                 return;
291             for (int i = 0; i < c.length; i++)
292                 processConnection(c[i]);
293         }
294     }
295
296     protected void setNote(int noteNumber) {
297         note = noteNumber;
298         tunedKey = tuning.getTuning(noteNumber) / 100.0;
299     }
300
301     protected void noteOn(int noteNumber, int velocity, int delay) {
302
303         sustain = false;
304         sostenuto = false;
305         portamento = false;
306
307         soundoff = false;
308         on = true;

```

```

309     active = true;
310     started = true;
311     // volume = velocity;
312
313     noteOn_noteNumber = noteNumber;
314     noteOn_velocity = velocity;
315     this.delay = delay;
316
317     lastMuteValue = 0;
318     lastSoloMuteValue = 0;
319
320     setNote(noteNumber);
321
322     if (performer.forcedKeynumber)
323         co_noteon_keynumber[0] = 0;
324     else
325         co_noteon_keynumber[0] = tunedKey * (1f / 128f);
326     if (performer.forcedVelocity)
327         co_noteon_velocity[0] = 0;
328     else
329         co_noteon_velocity[0] = velocity * (1f / 128f);
330     co_mixer_active[0] = 0;
331     co_mixer_gain[0] = 0;
332     co_mixer_pan[0] = 0;
333     co_mixer_balance[0] = 0;
334     co_mixer_reverb[0] = 0;
335     co_mixer_chorus[0] = 0;
336     co_osc_pitch[0] = 0;
337     co_filter_freq[0] = 0;
338     co_filter_q[0] = 0;
339     co_filter_type[0] = 0;
340     co_noteon_on[0] = 1;
341
342     eg.reset();
343     lfo.reset();
344     filter_left.reset();
345     filter_right.reset();
346
347     objects.put("master", synthesizer.getMainMixer().co_master);
348     objects.put("eg", eg);
349     objects.put("lfo", lfo);
350     objects.put("noteon", co_noteon);
351     objects.put("osc", co_osc);
352     objects.put("mixer", co_mixer);
353     objects.put("filter", co_filter);
354
355     connections = performer.connections;
356
357     if (connections_last == null
358         || connections_last.length < connections.length) {
359         connections_last = new double[connections.length];
360     }
361     if (connections_src == null
362         || connections_src.length < connections.length) {
363         connections_src = new double[connections.length][3];
364         connections_src_kc = new int[connections.length][2];
365     }
366     if (connections_dst == null
367         || connections_dst.length < connections.length) {
368         connections_dst = new double[connections.length][2];
369     }
370     for (int i = 0; i < connections.length; i++) {

```

```

371 ModelConnectionBlock conn = connections[i];
372 connections_last[i] = 0;
373 if (conn.getSources() != null) {
374     ModelSource[] srcs = conn.getSources();
375     if (connections_src[i] == null
376         || connections_src[i].length < srcs.length) {
377         connections_src[i] = new double[srcs.length][];
378         connections_src_kc[i] = new int[srcs.length];
379     }
380     double[][] src = connections_src[i];
381     int[] src_kc = connections_src_kc[i];
382     connections_src[i] = src;
383     for (int j = 0; j < srcs.length; j++) {
384         src_kc[j] = getValueKC(srcs[j].getIdentifier());
385         src[j] = getValue(srcs[j].getIdentifier());
386     }
387 }
388
389 if (conn.getDestination() != null)
390     connections_dst[i] = getValue(conn.getDestination()
391         .getIdentifier());
392 else
393     connections_dst[i] = null;
394 }
395
396 for (int i = 0; i < connections.length; i++)
397     processConnection(i);
398
399 if (extendedConnectionBlocks != null) {
400     for (ModelConnectionBlock connection: extendedConnectionBlocks) {
401         double value = 0;
402
403         if (softchannel.keybasedcontroller_active == null) {
404             for (ModelSource src: connection.getSources()) {
405                 double x = getValue(src.getIdentifier())[0];
406                 ModelTransform t = src.getTransform();
407                 if (t == null)
408                     value += x;
409                 else
410                     value += t.transform(x);
411             }
412         } else {
413             for (ModelSource src: connection.getSources()) {
414                 double x = getValue(src.getIdentifier())[0];
415                 x = processKeyBasedController(x,
416                     getValueKC(src.getIdentifier()));
417                 ModelTransform t = src.getTransform();
418                 if (t == null)
419                     value += x;
420                 else
421                     value += t.transform(x);
422             }
423         }
424
425         ModelDestination dest = connection.getDestination();
426         ModelTransform t = dest.getTransform();
427         if (t != null)
428             value = t.transform(value);
429         getValue(dest.getIdentifier())[0] += value;
430     }
431 }
432

```



```

433     eg.init(synthesizer);
434     lfo.init(synthesizer);
435
436 }
437
438 protected void setPolyPressure(int pressure) {
439     if(performer == null)
440         return;
441     int[] c = performer.midi_connections[2];
442     if (c == null)
443         return;
444     for (int i = 0; i < c.length; i++)
445         processConnection(c[i]);
446 }
447
448 protected void setChannelPressure(int pressure) {
449     if(performer == null)
450         return;
451     int[] c = performer.midi_connections[1];
452     if (c == null)
453         return;
454     for (int i = 0; i < c.length; i++)
455         processConnection(c[i]);
456 }
457
458 protected void controlChange(int controller, int value) {
459     if(performer == null)
460         return;
461     int[] c = performer.midi_ctrl_connections[controller];
462     if (c == null)
463         return;
464     for (int i = 0; i < c.length; i++)
465         processConnection(c[i]);
466 }
467
468 protected void nrpnChange(int controller, int value) {
469     if(performer == null)
470         return;
471     int[] c = performer.midi_nrpn_connections.get(controller);
472     if (c == null)
473         return;
474     for (int i = 0; i < c.length; i++)
475         processConnection(c[i]);
476 }
477
478 protected void rpnChange(int controller, int value) {
479     if(performer == null)
480         return;
481     int[] c = performer.midi_rpn_connections.get(controller);
482     if (c == null)
483         return;
484     for (int i = 0; i < c.length; i++)
485         processConnection(c[i]);
486 }
487
488 protected void setPitchBend(int bend) {
489     if(performer == null)
490         return;
491     int[] c = performer.midi_connections[0];
492     if (c == null)
493         return;
494     for (int i = 0; i < c.length; i++)

```

```

495         processConnection(c[i]);
496     }
497
498     protected void setMute(boolean mute) {
499         co_mixer_gain[0] -= lastMuteValue;
500         lastMuteValue = mute ? -960 : 0;
501         co_mixer_gain[0] += lastMuteValue;
502     }
503
504     protected void setSoloMute(boolean mute) {
505         co_mixer_gain[0] -= lastSoloMuteValue;
506         lastSoloMuteValue = mute ? -960 : 0;
507         co_mixer_gain[0] += lastSoloMuteValue;
508     }
509
510     protected void shutdown() {
511         if (co_noteon_on[0] < -0.5)
512             return;
513         on = false;
514
515         co_noteon_on[0] = -1;
516
517         if(performer == null)
518             return;
519         int[] c = performer.midi_connections[3];
520         if (c == null)
521             return;
522         for (int i = 0; i < c.length; i++)
523             processConnection(c[i]);
524     }
525
526     protected void soundOff() {
527         on = false;
528         soundoff = true;
529     }
530
531     protected void noteOff(int velocity) {
532         if (!on)
533             return;
534         on = false;
535
536         noteOff_velocity = velocity;
537
538         if (softchannel.sustain) {
539             sustain = true;
540             return;
541         }
542         if (sostenuto)
543             return;
544
545         co_noteon_on[0] = 0;
546
547         if(performer == null)
548             return;
549         int[] c = performer.midi_connections[3];
550         if (c == null)
551             return;
552         for (int i = 0; i < c.length; i++)
553             processConnection(c[i]);
554     }
555
556     protected void redamp() {

```

```

557     if (co_noteon_on[0] > 0.5)
558         return;
559     if (co_noteon_on[0] < -0.5)
560         return; // don't redamp notes in shutdown stage
561
562     sustain = true;
563     co_noteon_on[0] = 1;
564
565     if(performer == null)
566         return;
567     int[] c = performer.midi_connections[3];
568     if (c == null)
569         return;
570     for (int i = 0; i < c.length; i++)
571         processConnection(c[i]);
572 }
573
574 protected void processControlLogic() {
575     if (stopping) {
576         active = false;
577         stopping = false;
578         audiostarted = false;
579         instrument = null;
580         performer = null;
581         connections = null;
582         extendedConnectionBlocks = null;
583         channelmixer = null;
584         if (osc_stream != null)
585             try {
586                 osc_stream.close();
587             } catch (IOException e) {
588                 //e.printStackTrace();
589             }
590
591         if (stealer_channel != null) {
592             stealer_channel.initVoice(this, stealer_performer,
593                                     stealer_voiceID, stealer_noteNumber, stealer_velocity, 0,
594                                     stealer_extendedConnectionBlocks, stealer_channelmixer,
595                                     stealer_releaseTriggered);
596             stealer_releaseTriggered = false;
597             stealer_channel = null;
598             stealer_performer = null;
599             stealer_voiceID = -1;
600             stealer_noteNumber = 0;
601             stealer_velocity = 0;
602             stealer_extendedConnectionBlocks = null;
603             stealer_channelmixer = null;
604         }
605     }
606     if (started) {
607         audiostarted = true;
608
609         ModelOscillator osc = performer.oscillators[0];
610
611         osc_stream_off_transmitted = false;
612         if (osc instanceof ModelWavetable) {
613             try {
614                 resampler.open((ModelWavetable)osc,
615                               synthesizer.getFormat().getSampleRate());
616                 osc_stream = resampler;
617             } catch (IOException e) {
618                 //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 if (audiostarted) {
634     if (portamento) {
635         double note_delta = tunedKey - (co_noteon_keynumber[0] * 128);
636         double note_delta_a = Math.abs(note_delta);
637         if (note_delta_a < 0.0000000001) {
638             co_noteon_keynumber[0] = tunedKey * (1.0 / 128.0);
639             portamento = false;
640         } else {
641             if (note_delta_a > softchannel.portamento_time)
642                 note_delta = Math.signum(note_delta)
643                     * softchannel.portamento_time;
644             co_noteon_keynumber[0] += note_delta * (1.0 / 128.0);
645         }
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 * 0.7071067811865476f;
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;

```

```

743     out_mixer_end = co_mixer_active[0] < 0.5;
744
745     if (!on)
746         if (!osc_stream_off_transmitted) {
747             osc_stream_off_transmitted = true;
748             if (osc_stream != null)
749                 osc_stream.noteOff(noteOff_velocity);
750         }
751
752     }
753     if (started) {
754         last_out_mixer_left = out_mixer_left;
755         last_out_mixer_right = out_mixer_right;
756         last_out_mixer_effect1 = out_mixer_effect1;
757         last_out_mixer_effect2 = out_mixer_effect2;
758         started = false;
759     }
760
761 }
762
763 protected void mixAudioStream(SoftAudioBuffer in, SoftAudioBuffer out,
764     SoftAudioBuffer dout,
765     float amp_from, float amp_to) {
766     int bufferlen = in.getSize();
767     if (amp_from < 0.000000001 && amp_to < 0.000000001)
768         return;
769     if (dout != null && delay != 0)
770     {
771         if (amp_from == amp_to) {
772             float[] fout = out.array();
773             float[] fin = in.array();
774             int j = 0;
775             for (int i = delay; i < bufferlen; i++)
776                 fout[i] += fin[j++] * amp_to;
777             fout = dout.array();
778             for (int i = 0; i < delay; i++)
779                 fout[i] += fin[j++] * amp_to;
780         } else {
781             float amp = amp_from;
782             float amp_delta = (amp_to - amp_from) / bufferlen;
783             float[] fout = out.array();
784             float[] fin = in.array();
785             int j = 0;
786             for (int i = delay; i < bufferlen; i++) {
787                 amp += amp_delta;
788                 fout[i] += fin[j++] * amp;
789             }
790             fout = dout.array();
791             for (int i = 0; i < delay; i++) {
792                 amp += amp_delta;
793                 fout[i] += fin[j++] * amp;
794             }
795         }
796     }
797     else
798     {
799         if (amp_from == amp_to) {
800             float[] fout = out.array();
801             float[] fin = in.array();
802             for (int i = 0; i < bufferlen; i++)
803                 fout[i] += fin[i] * amp_to;
804         } else {

```

```

805         float amp = amp_from;
806         float amp_delta = (amp_to - amp_from) / bufferlen;
807         float[] fout = out.array();
808         float[] fin = in.array();
809         for (int i = 0; i < bufferlen; i++) {
810             amp += amp_delta;
811             fout[i] += fin[i] * amp;
812         }
813     }
814 }
815
816 }
817
818 protected void processAudioLogic(SoftAudioBuffer[] buffer) {
819     if (!audiostarted)
820         return;
821
822     int bufferlen = buffer[0].getSize();
823
824     try {
825         osc_buff[0] = buffer[SoftMainMixer.CHANNEL_LEFT_DRY].array();
826         if (nrofchannels != 1)
827             osc_buff[1] = buffer[SoftMainMixer.CHANNEL_RIGHT_DRY].array();
828         int ret = osc_stream.read(osc_buff, 0, bufferlen);
829         if (ret == -1) {
830             stopping = true;
831             return;
832         }
833         if (ret != bufferlen) {
834             Arrays.fill(osc_buff[0], ret, bufferlen, 0f);
835             if (nrofchannels != 1)
836                 Arrays.fill(osc_buff[1], ret, bufferlen, 0f);
837         }
838     } catch (IOException e) {
839         //e.printStackTrace();
840     }
841
842     SoftAudioBuffer left = buffer[SoftMainMixer.CHANNEL_LEFT];
843     SoftAudioBuffer right = buffer[SoftMainMixer.CHANNEL_RIGHT];
844     SoftAudioBuffer mono = buffer[SoftMainMixer.CHANNEL_MONO];
845     SoftAudioBuffer eff1 = buffer[SoftMainMixer.CHANNEL_EFFECT1];
846     SoftAudioBuffer eff2 = buffer[SoftMainMixer.CHANNEL_EFFECT2];
847
848     SoftAudioBuffer dleft = buffer[SoftMainMixer.CHANNEL_DELAY_LEFT];
849     SoftAudioBuffer dright = buffer[SoftMainMixer.CHANNEL_DELAY_RIGHT];
850     SoftAudioBuffer dmono = buffer[SoftMainMixer.CHANNEL_DELAY_MONO];
851     SoftAudioBuffer deff1 = buffer[SoftMainMixer.CHANNEL_DELAY_EFFECT1];
852     SoftAudioBuffer deff2 = buffer[SoftMainMixer.CHANNEL_DELAY_EFFECT2];
853
854     SoftAudioBuffer leftdry = buffer[SoftMainMixer.CHANNEL_LEFT_DRY];
855     SoftAudioBuffer rightdry = buffer[SoftMainMixer.CHANNEL_RIGHT_DRY];
856
857     if (osc_stream_nrofchannels == 1)
858         rightdry = null;
859
860     if (!Double.isInfinite(co_filter_freq[0])) {
861         filter_left.processAudio(leftdry);
862         if (rightdry != null)
863             filter_right.processAudio(rightdry);
864     }
865 }
866

```

```

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

```
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.io.BufferedInputStream;
28 import java.io.File;
29 import java.io.FileInputStream;
30 import java.io.IOException;
31 import java.io.InputStream;
32 import java.net.URL;
33 import java.util.HashMap;
34 import java.util.Map;
35
36 import javax.sound.sampled.AudioFileFormat;
37 import javax.sound.sampled.AudioFormat;
38 import javax.sound.sampled.AudioInputStream;
39 import javax.sound.sampled.AudioSystem;
40 import javax.sound.sampled.UnsupportedAudioFileException;
41 import javax.sound.sampled.AudioFormat.Encoding;
42 import javax.sound.sampled.spi.AudioFileReader;
43
44 /**
45  * WAVE file reader for files using format WAVE_FORMAT_EXTENSIBLE (0xFFFE).
46  *
47  * @author Karl Helgason
48  */
49 public class WaveExtensibleFileReader extends AudioFileReader {
50
51     static private class GUID {
52         long i1;
53
54         int s1;
55
56         int s2;
57
58         int x1;
59
60         int x2;
```

```

61     int x3;
62
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)

```

```

123         return false;
124     if (x2 != t.x2)
125         return false;
126     if (x3 != t.x3)
127         return false;
128     if (x4 != t.x4)
129         return false;
130     if (x5 != t.x5)
131         return false;
132     if (x6 != t.x6)
133         return false;
134     if (x7 != t.x7)
135         return false;
136     if (x8 != t.x8)
137         return false;
138     return true;
139 }
140
141 }
142
143 private static String[] channelnames = { "FL", "FR", "FC", "LF",
144     "BL",
145     "BR", // 5.1
146     "FLC", "FLR", "BC", "SL", "SR", "TC", "TFL", "TFC", "TFR", "TBL",
147     "TBC", "TBR" };
148
149 private static String[] allchannelnames = { "w1", "w2", "w3", "w4", "w5",
150     "w6", "w7", "w8", "w9", "w10", "w11", "w12", "w13", "w14", "w15",
151     "w16", "w17", "w18", "w19", "w20", "w21", "w22", "w23", "w24",
152     "w25", "w26", "w27", "w28", "w29", "w30", "w31", "w32", "w33",
153     "w34", "w35", "w36", "w37", "w38", "w39", "w40", "w41", "w42",
154     "w43", "w44", "w45", "w46", "w47", "w48", "w49", "w50", "w51",
155     "w52", "w53", "w54", "w55", "w56", "w57", "w58", "w59", "w60",
156     "w61", "w62", "w63", "w64" };
157
158 private static GUID SUBTYPE_PCM = new GUID(0x00000001, 0x0000, 0x0010,
159     0x80, 0x00, 0x00, 0xaa, 0x00, 0x38, 0x9b, 0x71);
160
161 private static GUID SUBTYPE_IEEE_FLOAT = new GUID(0x00000003, 0x0000,
162     0x0010, 0x80, 0x00, 0x00, 0xaa, 0x00, 0x38, 0x9b, 0x71);
163
164 private String decodeChannelMask(long channelmask) {
165     StringBuffer sb = new StringBuffer();
166     long m = 1;
167     for (int i = 0; i < allchannelnames.length; i++) {
168         if ((channelmask & m) != 0L) {
169             if (i < channelnames.length) {
170                 sb.append(channelnames[i] + "_");
171             } else {
172                 sb.append(allchannelnames[i] + "_");
173             }
174         }
175         m *= 2L;
176     }
177     if (sb.length() == 0)
178         return null;
179     return sb.substring(0, sb.length() - 1);
180 }
181
182
183 public AudioFileFormat getAudioFileFormat(InputStream stream)
184     throws UnsupportedAudioFileException, IOException {

```

```

185     stream.mark(200);
186     AudioFileFormat format;
187     try {
188         format = internal_getAudioFileFormat(stream);
189     } finally {
190         stream.reset();
191     }
192     return format;
193 }
194
195 private AudioFileFormat internal_getAudioFileFormat(InputStream stream)
196     throws UnsupportedAudioFileException, IOException {
197
198     RIFFReader riffiterator = new RIFFReader(stream);
199     if (!riffiterator.getFormat().equals("RIFF"))
200         throw new UnsupportedAudioFileException();
201     if (!riffiterator.getType().equals("WAVE"))
202         throw new UnsupportedAudioFileException();
203
204     boolean fmt_found = false;
205     boolean data_found = false;
206
207     int channels = 1;
208     long samplerate = 1;
209     // long framerate = 1;
210     int framesize = 1;
211     int bits = 1;
212     int validBitsPerSample = 1;
213     long channelMask = 0;
214     GUID subFormat = null;
215
216     while (riffiterator.hasNextChunk()) {
217         RIFFReader chunk = riffiterator.nextChunk();
218
219         if (chunk.getFormat().equals("fmt_")) {
220             fmt_found = true;
221
222             int format = chunk.readUnsignedShort();
223             if (format != 0xFFFE)
224                 throw new UnsupportedAudioFileException(); // WAVE_FORMAT_EXTENSIBLE
225             // only
226             channels = chunk.readUnsignedShort();
227             samplerate = chunk.readUnsignedInt();
228             /* framerate = */ chunk.readUnsignedInt();
229             framesize = chunk.readUnsignedShort();
230             bits = chunk.readUnsignedShort();
231             int cbSize = chunk.readUnsignedShort();
232             if (cbSize != 22)
233                 throw new UnsupportedAudioFileException();
234             validBitsPerSample = chunk.readUnsignedShort();
235             if (validBitsPerSample > bits)
236                 throw new UnsupportedAudioFileException();
237             channelMask = chunk.readUnsignedInt();
238             subFormat = GUID.read(chunk);
239
240         }
241         if (chunk.getFormat().equals("data")) {
242             data_found = true;
243             break;
244         }
245     }
246 }

```

```

247     if (!fmt_found)
248         throw new UnsupportedOperationException();
249     if (!data_found)
250         throw new UnsupportedOperationException();
251
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 UnsupportedOperationException();
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 UnsupportedOperationException, IOException {
287
288     AudioFileFormat format = getAudioFileFormat(stream);
289     RIFFReader riffiterator = new RIFFReader(stream);
290     if (!riffiterator.getFormat().equals("RIFF"))
291         throw new UnsupportedOperationException();
292     if (!riffiterator.getType().equals("WAVE"))
293         throw new UnsupportedOperationException();
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 UnsupportedOperationException();
302 }
303
304 public AudioFileFormat getAudioFileFormat(URL url)
305     throws UnsupportedOperationException, IOException {
306     InputStream stream = url.openStream();
307     AudioFileFormat format;
308     try {

```

```

309         format = getAudioFileFormat(new BufferedInputStream(stream));
310     } finally {
311         stream.close();
312     }
313     return format;
314 }
315
316 public AudioFileFormat getAudioFileFormat(File file)
317     throws UnsupportedAudioFileException, IOException {
318     InputStream stream = new FileInputStream(file);
319     AudioFileFormat format;
320     try {
321         format = getAudioFileFormat(new BufferedInputStream(stream));
322     } finally {
323         stream.close();
324     }
325     return format;
326 }
327
328 public AudioInputStream getAudioInputStream(URL url)
329     throws UnsupportedAudioFileException, IOException {
330     return getAudioInputStream(new BufferedInputStream(url.openStream()));
331 }
332
333 public AudioInputStream getAudioInputStream(File file)
334     throws UnsupportedAudioFileException, IOException {
335     return getAudioInputStream(new BufferedInputStream(new FileInputStream(
336         file)));
337 }
338
339 }

```

117 com/sun/media/sound/WaveFloatFileReader.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.io.BufferedInputStream;
28 import java.io.File;
29 import java.io.FileInputStream;
30 import java.io.IOException;
31 import java.io.InputStream;
32 import java.net.URL;
33
34 import javax.sound.sampled.AudioFileFormat;
35 import javax.sound.sampled.AudioFormat;
36 import javax.sound.sampled.AudioInputStream;
37 import javax.sound.sampled.AudioSystem;
38 import javax.sound.sampled.UnsupportedAudioFileException;
39 import javax.sound.sampled.spi.AudioFileReader;
40
41 /**
42 * Floating-point encoded (format 3) WAVE file loader.
43 *
44 * @author Karl Helgason
45 */
46 public class WaveFloatFileReader extends AudioFileReader {
47
48     public AudioFileFormat getAudioFileFormat(InputStream stream)
49         throws UnsupportedAudioFileException, IOException {
50
51         stream.mark(200);
52         AudioFileFormat format;
53         try {
54             format = internal_getAudioFileFormat(stream);
55         } finally {
56             stream.reset();
57         }
58         return format;
59     }
60 }
```

```

61 private AudioFileFormat internal_getAudioFileFormat(InputStream stream)
62     throws UnsupportedOperationException, IOException {
63
64     RIFFReader riffiterator = new RIFFReader(stream);
65     if (!riffiterator.getFormat().equals("RIFF"))
66         throw new UnsupportedOperationException();
67     if (!riffiterator.getType().equals("WAVE"))
68         throw new UnsupportedOperationException();
69
70     boolean fmt_found = false;
71     boolean data_found = false;
72
73     int channels = 1;
74     long samplerate = 1;
75     int framesize = 1;
76     int bits = 1;
77
78     while (riffiterator.hasNextChunk()) {
79         RIFFReader chunk = riffiterator.nextChunk();
80
81         if (chunk.getFormat().equals("fmt_")) {
82             fmt_found = true;
83
84             int format = chunk.readUnsignedShort();
85             if (format != 3) // WAVE_FORMAT_IEEE_FLOAT only
86                 throw new UnsupportedOperationException();
87             channels = chunk.readUnsignedShort();
88             samplerate = chunk.readUnsignedInt();
89             /* framerate = */ chunk.readUnsignedInt();
90             framesize = chunk.readUnsignedShort();
91             bits = chunk.readUnsignedShort();
92         }
93         if (chunk.getFormat().equals("data")) {
94             data_found = true;
95             break;
96         }
97     }
98
99     if (!fmt_found)
100         throw new UnsupportedOperationException();
101     if (!data_found)
102         throw new UnsupportedOperationException();
103
104     AudioFormat audioformat = new AudioFormat(
105         AudioFloatConverter.PCM_FLOAT, samplerate, bits, channels,
106         framesize, samplerate, false);
107     AudioFileFormat fileformat = new AudioFileFormat(
108         AudioFileFormat.Type.WAVE, audioformat,
109         AudioSystem.NOT_SPECIFIED);
110     return fileformat;
111 }
112
113 public AudioInputStream getAudioInputStream(InputStream stream)
114     throws UnsupportedOperationException, IOException {
115
116     AudioFileFormat format = getAudioFileFormat(stream);
117     RIFFReader riffiterator = new RIFFReader(stream);
118     if (!riffiterator.getFormat().equals("RIFF"))
119         throw new UnsupportedOperationException();
120     if (!riffiterator.getType().equals("WAVE"))
121         throw new UnsupportedOperationException();
122     while (riffiterator.hasNextChunk()) {

```



```

123         RIFFReader chunk = riffiterator.nextChunk();
124         if (chunk.getFormat().equals("data")) {
125             return new AudioInputStream(chunk, format.getFormat(),
126                 chunk.getSize());
127         }
128     }
129     throw new UnsupportedAudioFileException();
130 }
131
132 public AudioFileFormat getAudioFileFormat(URL url)
133     throws UnsupportedAudioFileException, IOException {
134     InputStream stream = url.openStream();
135     AudioFileFormat format;
136     try {
137         format = getAudioFileFormat(new BufferedInputStream(stream));
138     } finally {
139         stream.close();
140     }
141     return format;
142 }
143
144 public AudioFileFormat getAudioFileFormat(File file)
145     throws UnsupportedAudioFileException, IOException {
146     InputStream stream = new FileInputStream(file);
147     AudioFileFormat format;
148     try {
149         format = getAudioFileFormat(new BufferedInputStream(stream));
150     } finally {
151         stream.close();
152     }
153     return format;
154 }
155
156 public AudioInputStream getAudioInputStream(URL url)
157     throws UnsupportedAudioFileException, IOException {
158     return getAudioInputStream(new BufferedInputStream(url.openStream()));
159 }
160
161 public AudioInputStream getAudioInputStream(File file)
162     throws UnsupportedAudioFileException, IOException {
163     return getAudioInputStream(new BufferedInputStream(new FileInputStream(
164         file)));
165 }
166 }

```

118 com/sun/media/sound/WaveFloatFileWriter.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.File;
28 import java.io.IOException;
29 import java.io.OutputStream;
30
31 import javax.sound.sampled.AudioFileFormat;
32 import javax.sound.sampled.AudioFormat;
33 import javax.sound.sampled.AudioInputStream;
34 import javax.sound.sampled.AudioSystem;
35 import javax.sound.sampled.AudioFileFormat.Type;
36 import javax.sound.sampled.spi.AudioFileWriter;
37
38 /**
39 * Floating-point encoded (format 3) WAVE file writer.
40 *
41 * @author Karl Helgason
42 */
43 public class WaveFloatFileWriter extends AudioFileWriter {
44
45     public Type[] getAudioFileTypes() {
46         return new Type[] { Type.WAVE };
47     }
48
49     public Type[] getAudioFileTypes(AudioInputStream stream) {
50
51         if (!stream.getFormat().getEncoding().equals(
52             AudioFloatConverter.PCM_FLOAT))
53             return new Type[0];
54         return new Type[] { Type.WAVE };
55     }
56
57     private void checkFormat(AudioFileFormat.Type type, AudioInputStream stream) {
58         if (!Type.WAVE.equals(type))
59             throw new IllegalArgumentException("File_type_" + type
60                 + "_not_supported.");
61     }
62 }
```

```

61     if (!stream.getFormat().getEncoding().equals(
62         AudioFloatConverter.PCM_FLOAT))
63         throw new IllegalArgumentException("File_format_"
64             + stream.getFormat() + "_not_supported.");
65     }
66
67     public void write(AudioInputStream stream, RIFFWriter writer)
68         throws IOException {
69
70         RIFFWriter fmt_chunk = writer.writeChunk("fmt_");
71
72         AudioFormat format = stream.getFormat();
73         fmt_chunk.writeUnsignedShort(3); // WAVE_FORMAT_IEEE_FLOAT
74         fmt_chunk.writeUnsignedShort(format.getChannels());
75         fmt_chunk.writeUnsignedInt((int) format.getSampleRate());
76         fmt_chunk.writeUnsignedInt(((int) format.getFrameRate())
77             * format.getFrameSize());
78         fmt_chunk.writeUnsignedShort(format.getFrameSize());
79         fmt_chunk.writeUnsignedShort(format.getSampleSizeInBits());
80         fmt_chunk.close();
81         RIFFWriter data_chunk = writer.writeChunk("data");
82         byte[] buff = new byte[1024];
83         int len;
84         while ((len = stream.read(buff, 0, buff.length)) != -1)
85             data_chunk.write(buff, 0, len);
86         data_chunk.close();
87     }
88
89     private static class NoCloseOutputStream extends OutputStream {
90         OutputStream out;
91
92         public NoCloseOutputStream(OutputStream out) {
93             this.out = out;
94         }
95
96         public void write(int b) throws IOException {
97             out.write(b);
98         }
99
100        public void flush() throws IOException {
101            out.flush();
102        }
103
104        public void write(byte[] b, int off, int len) throws IOException {
105            out.write(b, off, len);
106        }
107
108        public void write(byte[] b) throws IOException {
109            out.write(b);
110        }
111    }
112
113    private AudioInputStream toLittleEndian(AudioInputStream ais) {
114        AudioFormat format = ais.getFormat();
115        AudioFormat targetFormat = new AudioFormat(format.getEncoding(), format
116            .getSampleRate(), format.getSampleSizeInBits(), format
117            .getChannels(), format.getFrameSize(), format.getFrameRate(),
118            false);
119        return AudioSystem.getAudioInputStream(targetFormat, ais);
120    }
121
122    public int write(AudioInputStream stream, Type fileType, OutputStream out)

```

```

123         throws IOException {
124
125         checkFormat(fileType, stream);
126         if (stream.getFormat().isBigEndian())
127             stream = toLittleEndian(stream);
128         RIFFWriter writer = new RIFFWriter(new NoCloseOutputStream(out), "WAVE");
129         write(stream, writer);
130         int fpointer = (int) writer.getFilePointer();
131         writer.close();
132         return fpointer;
133     }
134
135     public int write(AudioInputStream stream, Type fileType, File out)
136         throws IOException {
137         checkFormat(fileType, stream);
138         if (stream.getFormat().isBigEndian())
139             stream = toLittleEndian(stream);
140         RIFFWriter writer = new RIFFWriter(out, "WAVE");
141         write(stream, writer);
142         int fpointer = (int) writer.getFilePointer();
143         writer.close();
144         return fpointer;
145     }
146
147 }

```

119 simplemidiplayer/ConfigDialog.java

```
1  /*
2   * Copyright (c) 2007 by Karl Helgason
3   * All rights reserved.
4   *
5   * Redistribution and use in source and binary forms, with or without
6   * modification, are permitted provided that the following conditions
7   * are met:
8   *
9   * - Redistributions of source code must retain the above copyright notice,
10  *   this list of conditions and the following disclaimer.
11  * - Redistributions in binary form must reproduce the above copyright
12  *   notice, this list of conditions and the following disclaimer in the
13  *   documentation and/or other materials provided with the distribution.
14  *
15  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
16  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
17  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
18  * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
19  * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
20  * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
21  * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
22  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
23  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
24  * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
25  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
26  * OF THE POSSIBILITY OF SUCH DAMAGE.
27  */
28
29 package simplemidiplayer;
30
31 import java.awt.BorderLayout;
32 import java.awt.GridBagConstraints;
33 import java.awt.GridBagLayout;
34 import java.awt.Insets;
35 import java.awt.event.ActionEvent;
36 import java.awt.event.ActionListener;
37 import java.awt.event.KeyEvent;
38 import java.util.ArrayList;
39 import java.util.Properties;
40
41 import javax.sound.sampled.AudioSystem;
42 import javax.sound.sampled.Line;
43 import javax.sound.sampled.Mixer;
44 import javax.swing.BorderFactory;
45 import javax.swing.JButton;
46 import javax.swing.JComboBox;
47 import javax.swing.JComponent;
48 import javax.swing.JDialog;
49 import javax.swing.JFrame;
50 import javax.swing.JLabel;
51 import javax.swing.JPanel;
52 import javax.swing.KeyStroke;
53
54 public class ConfigDialog extends JDialog {
55
56     private static final long serialVersionUID = 1L;
57
58     boolean isok = false;
59
60     private String getValueFromList(JComboBox box) {
```

```

61     if (box.isEditable())
62         return box.getEditor().getItem().toString();
63     return box.getSelectedItem().toString();
64 }
65
66 private void selectValueInList(JComboBox box, String value) {
67     if (value == null)
68         return;
69     if (box.isEditable()) {
70         box.getEditor().setItem(value);
71     } else {
72         for (int i = 0; i < box.getItemCount(); i++) {
73             if (box.getItemAt(i).equals(value)) {
74                 box.setSelectedIndex(i);
75                 return;
76             }
77         }
78     }
79 }
80
81 public boolean isOK() {
82     return isok;
83 }
84
85 public ConfigDialog(JFrame parent) {
86     super(parent);
87     setSize(550, 400);
88     // setLocationByPlatform(true);
89     setModal(true);
90     setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
91     setTitle("Gervill_~MIDI_Player_~Config");
92
93     ArrayList<String> dev_list = new ArrayList<String>();
94     dev_list.add("(default)");
95     for (Mixer.Info info : AudioSystem.getMixerInfo()) {
96         Mixer mixer = AudioSystem.getMixer(info);
97         boolean hassrcline = false;
98         for (Line.Info linfo : mixer.getSourceLineInfo())
99             if (linfo instanceof javax.sound.sampled.DataLine.Info)
100                 hassrcline = true;
101         if (hassrcline) {
102             dev_list.add(info.getName());
103         }
104     }
105
106     String[] devlist = new String[dev_list.size()];
107     dev_list.toArray(devlist);
108
109     JPanel panel = new JPanel();
110     panel.setLayout(new BorderLayout());
111     setContentPane(panel);
112
113     JPanel optpanel = new JPanel();
114     optpanel.setBorder(BorderFactory.createEmptyBorder(3, 3, 3, 3));
115     panel.add(optpanel);
116     optpanel.setLayout(new GridBagLayout());
117
118     GridBagConstraints c = new GridBagConstraints();
119     c.insets = new Insets(3, 3, 3, 3);
120     c.anchor = GridBagConstraints.WEST;
121
122     final JComboBox co_devname = new JComboBox(devlist);

```

```

123 final JComboBox co_samplerate = new JComboBox(new String[] { "44100",
124     "22050", "11025" });
125 co_samplerate.setSelectedIndex(0);
126 co_samplerate.setEditable(true);
127 final JComboBox co_channels = new JComboBox(new String[] { "1", "2" });
128 co_channels.setSelectedIndex(1);
129 final JComboBox co_bits = new JComboBox(new String[] { "8", "16" });
130 co_bits.setSelectedIndex(1);
131 final JComboBox co_latency = new JComboBox(new String[] { "100", "200",
132     "400", "800" });
133 co_latency.setSelectedIndex(2);
134 co_latency.setEditable(true);
135 final JComboBox co_polyphony = new JComboBox(new String[] { "32", "64",
136     "96", "128", "256" });
137 co_polyphony.setSelectedIndex(1);
138 co_polyphony.setEditable(true);
139 final JComboBox co_interp = new JComboBox(new String[] { "linear",
140     "cubic", "sinc", "point" });
141 co_interp.setSelectedIndex(0);
142
143     final JComboBox co_largemode = new JComboBox(new String[] { "true",
144         "false" });
145     co_largemode.setSelectedIndex(1);
146
147 c.gridx = 0;
148 c.gridx = 0;
149 optpanel.add(new JLabel("Device_name:"), c);
150 c.gridx = 0;
151 c.gridx = 1;
152 optpanel.add(co_devname, c);
153 c.gridx = 1;
154 c.gridx = 0;
155 optpanel.add(new JLabel("Sample_rate_(Hz):"), c);
156 c.gridx = 1;
157 c.gridx = 1;
158 optpanel.add(co_samplerate, c);
159 c.gridx = 2;
160 c.gridx = 0;
161 optpanel.add(new JLabel("Channels:"), c);
162 c.gridx = 2;
163 c.gridx = 1;
164 optpanel.add(co_channels, c);
165 c.gridx = 3;
166 c.gridx = 0;
167 optpanel.add(new JLabel("Bits:"), c);
168 c.gridx = 3;
169 c.gridx = 1;
170 optpanel.add(co_bits, c);
171 c.gridx = 4;
172 c.gridx = 0;
173 optpanel.add(new JLabel("Latency_(msec):"), c);
174 c.gridx = 4;
175 c.gridx = 1;
176 optpanel.add(co_latency, c);
177 c.gridx = 5;
178 c.gridx = 0;
179 optpanel.add(new JLabel("Max_polyphony:"), c);
180 c.gridx = 5;
181 c.gridx = 1;
182 optpanel.add(co_polyphony, c);
183 c.gridx = 6;
184 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`

```
1  /*
2   * Copyright (c) 2007 by Karl Helgason
3   * All rights reserved.
4   *
5   * Redistribution and use in source and binary forms, with or without
6   * modification, are permitted provided that the following conditions
7   * are met:
8   *
9   * - Redistributions of source code must retain the above copyright notice,
10  *   this list of conditions and the following disclaimer.
11  * - Redistributions in binary form must reproduce the above copyright
12  *   notice, this list of conditions and the following disclaimer in the
13  *   documentation and/or other materials provided with the distribution.
14  *
15  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
16  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
17  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
18  * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
19  * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
20  * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
21  * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
22  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
23  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
24  * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
25  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
26  * OF THE POSSIBILITY OF SUCH DAMAGE.
27  */
28
29 package simplemidiplayer;
30
31 import java.awt.BorderLayout;
32 import java.awt.Font;
33 import java.awt.event.ActionEvent;
34 import java.awt.event.ActionListener;
35 import java.awt.event.KeyEvent;
36 import java.io.UnsupportedEncodingException;
37
38 import javax.sound.midi.Instrument;
39 import javax.sound.midi.MetaMessage;
40 import javax.sound.midi.MidiChannel;
41 import javax.sound.midi.MidiEvent;
42 import javax.sound.midi.Patch;
43 import javax.sound.midi.Sequence;
44 import javax.sound.midi.ShortMessage;
45 import javax.sound.midi.Soundbank;
46 import javax.sound.midi.Track;
47 import javax.sound.midi.VoiceStatus;
48 import javax.swing.BorderFactory;
49 import javax.swing.ImageIcon;
50 import javax.swing.JComponent;
51 import javax.swing.JFrame;
52 import javax.swing.JLabel;
53 import javax.swing.JPanel;
54 import javax.swing.JScrollPane;
55 import javax.swing.JTabbedPane;
56 import javax.swing.JTable;
57 import javax.swing.KeyStroke;
58 import javax.swing.table.DefaultTableModel;
59
60 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);

```

```

185 chtable.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
186 chtable.getColumnModel().getColumn(0).setPreferredWidth(65);
187 chtable.getColumnModel().getColumn(1).setPreferredWidth(100);
188 chtable.getColumnModel().getColumn(2).setPreferredWidth(65);
189 chtable.getColumnModel().getColumn(3).setPreferredWidth(65);
190 chtable.getColumnModel().getColumn(4).setPreferredWidth(65);
191 chtable.getColumnModel().getColumn(5).setPreferredWidth(65);
192 chtable.getColumnModel().getColumn(6).setPreferredWidth(65);
193 chmodel.addRow(new Object[] { "1", "", 0, 100, 0, 64, 0 });
194 chmodel.addRow(new Object[] { "2", "", 0, 100, 0, 64, 0 });
195 chmodel.addRow(new Object[] { "3", "", 0, 100, 0, 64, 0 });
196 chmodel.addRow(new Object[] { "4", "", 0, 100, 0, 64, 0 });
197 chmodel.addRow(new Object[] { "5", "", 0, 100, 0, 64, 0 });
198 chmodel.addRow(new Object[] { "6", "", 0, 100, 0, 64, 0 });
199 chmodel.addRow(new Object[] { "7", "", 0, 100, 0, 64, 0 });
200 chmodel.addRow(new Object[] { "8", "", 0, 100, 0, 64, 0 });
201 chmodel.addRow(new Object[] { "9", "", 0, 100, 0, 64, 0 });
202 chmodel.addRow(new Object[] { "10", "", 0, 100, 0, 64, 0 });
203 chmodel.addRow(new Object[] { "11", "", 0, 100, 0, 64, 0 });
204 chmodel.addRow(new Object[] { "12", "", 0, 100, 0, 64, 0 });
205 chmodel.addRow(new Object[] { "13", "", 0, 100, 0, 64, 0 });
206 chmodel.addRow(new Object[] { "14", "", 0, 100, 0, 64, 0 });
207 chmodel.addRow(new Object[] { "15", "", 0, 100, 0, 64, 0 });
208 chmodel.addRow(new Object[] { "16", "", 0, 100, 0, 64, 0 });
209
210 chtab.add(new JScrollPane(chtable));
211 chtab.setOpaque(false);
212 tabs.addTab("Channels", chtab);
213
214 voctab = new JPanel();
215 voctab.setLayout(new BorderLayout());
216 JTable voctable = new JTable();
217
218 vocmodel = new DefaultTableModel() {
219     private static final long serialVersionUID = 1L;
220
221     public boolean isCellEditable(int row, int column) {
222         return false;
223     }
224 };
225 vocmodel.addColumn("Active");
226 vocmodel.addColumn("Channel");
227 vocmodel.addColumn("Bank");
228 vocmodel.addColumn("Program");
229 vocmodel.addColumn("Note");
230 vocmodel.addColumn("Volume");
231 voctable.setModel(vocmodel);
232 voctable.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
233 voctable.getColumnModel().getColumn(0).setPreferredWidth(65);
234 voctable.getColumnModel().getColumn(1).setPreferredWidth(64);
235 voctable.getColumnModel().getColumn(2).setPreferredWidth(65);
236 voctable.getColumnModel().getColumn(3).setPreferredWidth(65);
237 voctable.getColumnModel().getColumn(4).setPreferredWidth(65);
238 voctable.getColumnModel().getColumn(5).setPreferredWidth(65);
239
240 voctab.add(new JScrollPane(voctable));
241 voctab.setOpaque(false);
242 tabs.addTab("Voices", voctab);
243
244 ImageIcon swan = new javax.swing.ImageIcon(getClass().getResource(
245     "/simplemidiplayer/swan.png"));
246 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`

```
1  /*
2   * Copyright (c) 2007 by Karl Helgason
3   * All rights reserved.
4   *
5   * Redistribution and use in source and binary forms, with or without
6   * modification, are permitted provided that the following conditions
7   * are met:
8   *
9   * - Redistributions of source code must retain the above copyright notice,
10  *   this list of conditions and the following disclaimer.
11  * - Redistributions in binary form must reproduce the above copyright
12  *   notice, this list of conditions and the following disclaimer in the
13  *   documentation and/or other materials provided with the distribution.
14  *
15  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
16  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
17  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
18  * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
19  * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
20  * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
21  * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
22  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
23  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
24  * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
25  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
26  * OF THE POSSIBILITY OF SUCH DAMAGE.
27  */
28
29 package simplemidiplayer;
30
31 import java.awt.Dimension;
32 import java.awt.FlowLayout;
33 import java.awt.Font;
34 import java.awt.Graphics;
35 import java.awt.GraphicsConfiguration;
36 import java.awt.GraphicsEnvironment;
37 import java.awt.Insets;
38 import java.awt.Point;
39 import java.awt.Rectangle;
40 import java.awt.Toolkit;
41 import java.awt.Window;
42 import java.awt.datatransfer.DataFlavor;
43 import java.awt.datatransfer.Transferable;
44 import java.awt.datatransfer.UnsupportedFlavorException;
45 import java.awt.event.ActionEvent;
46 import java.awt.event.ActionListener;
47 import java.awt.event.WindowAdapter;
48 import java.awt.event.WindowEvent;
49 import java.io.File;
50 import java.io.FileInputStream;
51 import java.io.FileOutputStream;
52 import java.io.IOException;
53 import java.util.HashMap;
54 import java.util.List;
55 import java.util.Map;
56 import java.util.Properties;
57
58 import javax.sound.midi.InvalidMidiDataException;
59 import javax.sound.midi.MidiChannel;
60 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    private static final long serialVersionUID = 1L;
112
113    public static void main(String[] args) {
114        if (!configExists()) {
115            ConfigDialog cd = new ConfigDialog(null);
116            cd.setVisible(true);
117            if (!cd.isOK())
118                return;
119        }
120        new SimpleMidiPlayer().setVisible(true);
121    }
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 Properties p = getConfig();
186 Map<String, Object> ainfo = new HashMap<String, Object>();
187
188 try {
189
190     format = new AudioFormat(Float.parseFloat(p
191         .getProperty("samplerate", "44100")), Integer
192         .parseInt(p.getProperty("bits", "16")), Integer
193         .parseInt(p.getProperty("channels", "2")), true, false);
194
195     int latency = Integer.parseInt(p.getProperty("latency", "200")) * 1000;
196
197     String devname = p.getProperty("devicename");
198     if (devname != null) {
199         Mixer.Info selinfo = null;
200         for (Mixer.Info info : AudioSystem.getMixerInfo()) {
201             Mixer mixer = AudioSystem.getMixer(info);
202             boolean hassrcline = false;
203             for (Line.Info linfo : mixer.getSourceLineInfo())
204                 if (linfo instanceof javax.sound.sampled.DataLine.Info)
205                     hassrcline = true;
206             if (hassrcline) {
207                 if (info.getName().equals(devname)) {
208                     selinfo = info;
209                     break;
210                 }
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 public void initMIDI_inThread() {
292     synth_loaded = false;
293     new Thread() {
294         public void run() {
295             initMIDI();
296         }
297     }.start();
298
299 }
300
301 public void closeMIDI() {
302     if (synth_loaded) {
303         seqr.close();
304         softsynth.close();
305         if (line != null) {
306             line.close();
307             line = null;
308         }

```

```

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() / seqr.getResolution();
353                 long b = seqr.getTickLength() / seqr.getResolution();
354                 if (seqr.isRunning())
355                     line3 = "PLAY " + a + " of " + b;
356                 else
357                     line3 = "STOP " + a + " of " + b;
358             } else {
359                 if (seq_errmsg != null)
360                     line3 = seq_errmsg;
361                 else
362                     line3 = seqfile.getName();
363                 if (line3.length() > 31)
364                     line3 = line3.substring(0, 31);
365             }
366         }
367     }
368     displayLab.setText("<html><body>" + line1 + "<br>" + line2 + "<br>"
369         + line3);
370 }

```

```

371 }
372
373 JFileChooser loadseq;
374
375 JFileChooser loadsndbk;
376
377 Thread actdisplay;
378
379 boolean player_running = true;
380
381 private static String CONFIG_FILE_NAME = "SimpleMidiPlayer.xml";
382
383 private static File userDir = new File(System.getProperty("user.home"),
384     ".gervill");
385
386 private static File configFile = new File(userDir, CONFIG_FILE_NAME);
387
388 private static Properties configp = null;
389
390 public static void centerWindow(Window w) {
391     Rectangle windowSize;
392     // Insets windowInsets;
393
394     Toolkit toolkit = Toolkit.getDefaultToolkit();
395     GraphicsEnvironment ge = java.awt.GraphicsEnvironment
396         .getLocalGraphicsEnvironment();
397     GraphicsConfiguration gc = ge.getDefaultScreenDevice()
398         .getDefaultConfiguration();
399     if (gc == null)
400         gc = w.getGraphicsConfiguration();
401
402     if (gc != null) {
403         windowSize = gc.getBounds();
404     } else {
405         windowSize = new java.awt.Rectangle(toolkit.getScreenSize());
406     }
407
408     Dimension size = w.getSize();
409     Point parent_loc = w.getLocation();
410     w.setLocation(parent_loc.x + windowSize.width / 2 - (size.width / 2),
411         parent_loc.y + windowSize.height / 2 - (size.height / 2));
412 }
413
414
415 public static boolean configExists() {
416     synchronized (configFile) {
417         return configFile.exists();
418     }
419 }
420
421 public static Properties getConfig() {
422     synchronized (configFile) {
423         if (configp != null) {
424             Properties p = new Properties();
425             p.putAll(configp);
426             return p;
427         }
428         Properties p = new Properties();
429         if (configFile.exists()) {
430             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) {

```



```

495     try {
496         sbk_errmsg = null;
497         Soundbank newsbk = MidiSystem.getSoundbank(newsbkfile);
498         if (sbk != null)
499             softsynth.unloadAllInstruments(sbk);
500         sbkfile = newsbkfile;
501         sbk = newsbk;
502         softsynth.loadAllInstruments(sbk);
503     } catch (Throwable e1) {
504         sbk_errmsg = e1.toString();
505     }
506
507 }
508
509 public SimpleMidiPlayer() {
510
511     loadseq = new JFileChooser();
512     loadseq.setDialogTitle("Load_MIDI_Sequence");
513     loadseq.setFileFilter(new FileFilter() {
514         public boolean accept(File f) {
515             if (!f.isFile())
516                 return true;
517             return f.getName().toLowerCase().endsWith(".mid");
518         }
519
520         public String getDescription() {
521             return "MIDI_Sequence";
522         }
523     });
524     loadsndbk = new JFileChooser();
525     loadsndbk.setFileFilter(new FileFilter() {
526         public boolean accept(File f) {
527             if (!f.isFile())
528                 return true;
529             String name = f.getName().toLowerCase();
530             if (name.endsWith(".sf2"))
531                 return true;
532             if (name.endsWith(".dls"))
533                 return true;
534             if (name.endsWith(".pat"))
535                 return true;
536             if (name.endsWith(".cfg"))
537                 return true;
538             if (name.endsWith(".wav"))
539                 return true;
540             if (name.endsWith(".au"))
541                 return true;
542             if (name.endsWith(".aif"))
543                 return true;
544             return false;
545         }
546
547         public String getDescription() {
548             return "SoundBank_(.sf2,.dls,.pat,.cfg,.wav,.au,.aif)";
549         }
550     });
551
552     // setLocationByPlatform(true);
553     setResizable(false);
554     setTitle("Gervill_-_MIDI_Player");
555     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
556

```

```

557 addWindowListener(new WindowAdapter() {
558     public void windowClosing(WindowEvent e) {
559         closeMIDI();
560         player_running = false;
561         try {
562             actdisplay.join(1000);
563         } catch (InterruptedException e1) {
564             e1.printStackTrace();
565         }
566     }
567 });
568
569 infoframe = new InfoFrame(this);
570
571 actdisplay = new Thread() {
572     public void run() {
573         boolean ok = true;
574         while (ok) {
575             synchronized (SimpleMidiPlayer.this) {
576                 ok = player_running;
577             }
578             SwingUtilities.invokeLater(new Runnable() {
579                 public void run() {
580                     updateDisplay();
581                     infoframe.updateDisplay();
582                 }
583             });
584             try {
585                 Thread.sleep(100);
586             } catch (InterruptedException e) {
587                 e.printStackTrace();
588                 return;
589             }
590         }
591     }
592 };
593 actdisplay.start();
594
595 initMIDI_inThread();
596
597 ImageIcon backgr = new javax.swing.ImageIcon(getClass().getResource(
598     "/simplemidiplayer/backgr.png"));
599 ImageIcon swan = new javax.swing.ImageIcon(getClass().getResource(
600     "/simplemidiplayer/swan.png"));
601 setIconImage(swan.getImage());
602
603 JPanel panel = new ImagePanel(backgr);
604 Dimension size = new Dimension(443, 125);
605 panel.setPreferredSize(size);
606 panel.setMinimumSize(size);
607 panel.setLayout(null);
608
609 TransferHandler thandler = new TransferHandler() {
610     private static final long serialVersionUID = 1L;
611
612     public boolean canImport(JComponent comp,
613         DataFlavor[] transferFlavors) {
614
615         for (int i = 0; i < transferFlavors.length; i++) {
616             if (transferFlavors[i]
617                 .equals(DataFlavor.javaFileListFlavor)) {
618                 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 config.addActionListener(new ActionListener() {
744     public void actionPerformed(ActionEvent e) {
745         if (!synth_loaded)
746             return;
747
748         ConfigDialog cd = new ConfigDialog(SimpleMidiPlayer.this);
749         cd.setVisible(true);
750         if (cd.isOK()) {
751             Sequence pseq = seqr.getSequence();
752             long ptick = seqr.getTickPosition();
753             boolean pruning = seqr.isRunning();
754             seqr.stop();
755             softsynth.close();
756             if (synthmixer != null) {
757                 synthmixer.close();
758                 synthmixer = null;
759             }
760             initMIDI();
761             if (pseq != null) {
762                 try {
763                     seqr.setSequence(pseq);
764                 } catch (InvalidMidiDataException e1) {
765                     e1.printStackTrace();
766                 }
767             }
768             seqr.setTickPosition(ptick);
769             if (pruning) {
770                 seqr.start();
771             }
772         }
773     }
774 }
775 });
776
777 load.addActionListener(new ActionListener() {
778     public void actionPerformed(ActionEvent e) {
779         if (!synth_loaded)
780             return;
781         loadmenu.show(load, 0, 0);
782     }
783 });
784
785 info.addActionListener(new ActionListener() {
786     public void actionPerformed(ActionEvent e) {
787         infoframe.setVisible(!infoframe.isVisible());
788     }
789 });
790
791 play.addActionListener(new ActionListener() {
792     public void actionPerformed(ActionEvent e) {
793         if (!synth_loaded)
794             return;
795         if (seq == null)
796             return;
797         seqr.start();
798     }
799 });
800
801 stop.addActionListener(new ActionListener() {
802     public void actionPerformed(ActionEvent e) {

```

```

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 }

```