Anhang A

Material zu „Studioluft“

Abbildungen

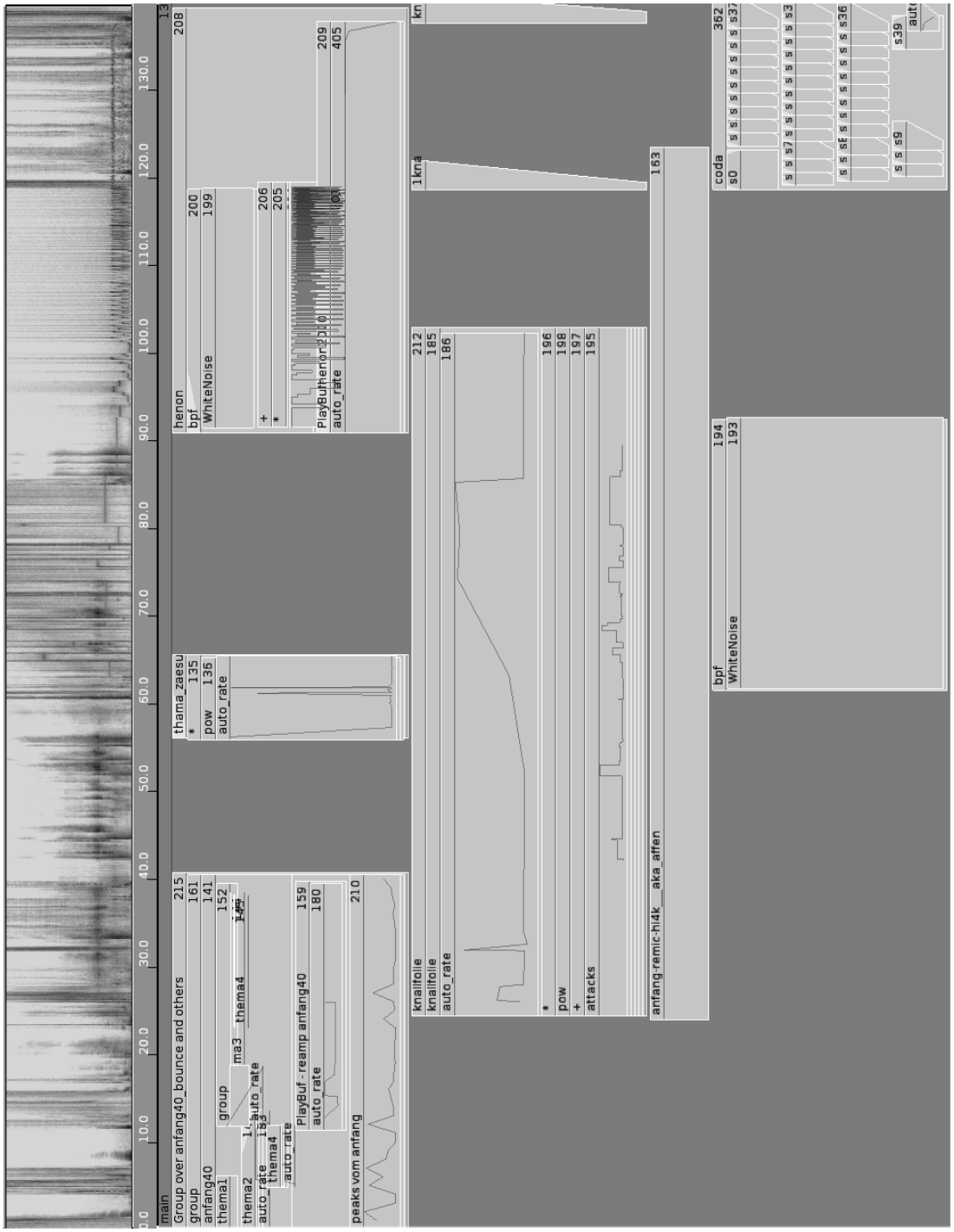

Abbildung A.1: Die Partitur von Studioluft mit Spektrogramm

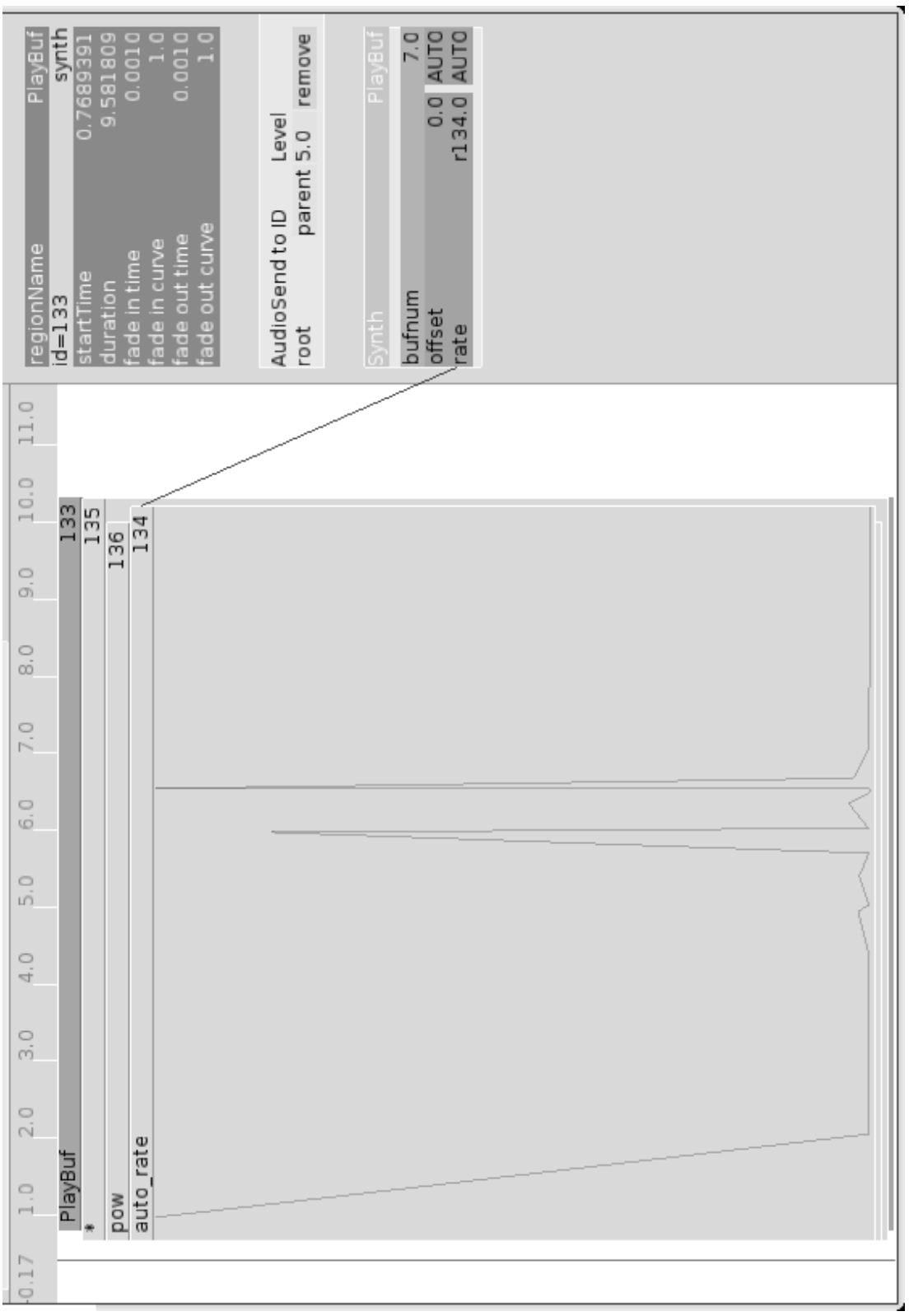

Abbildung A.2: Thema: Buffer 7 wird mit variabler Rate abgespielt

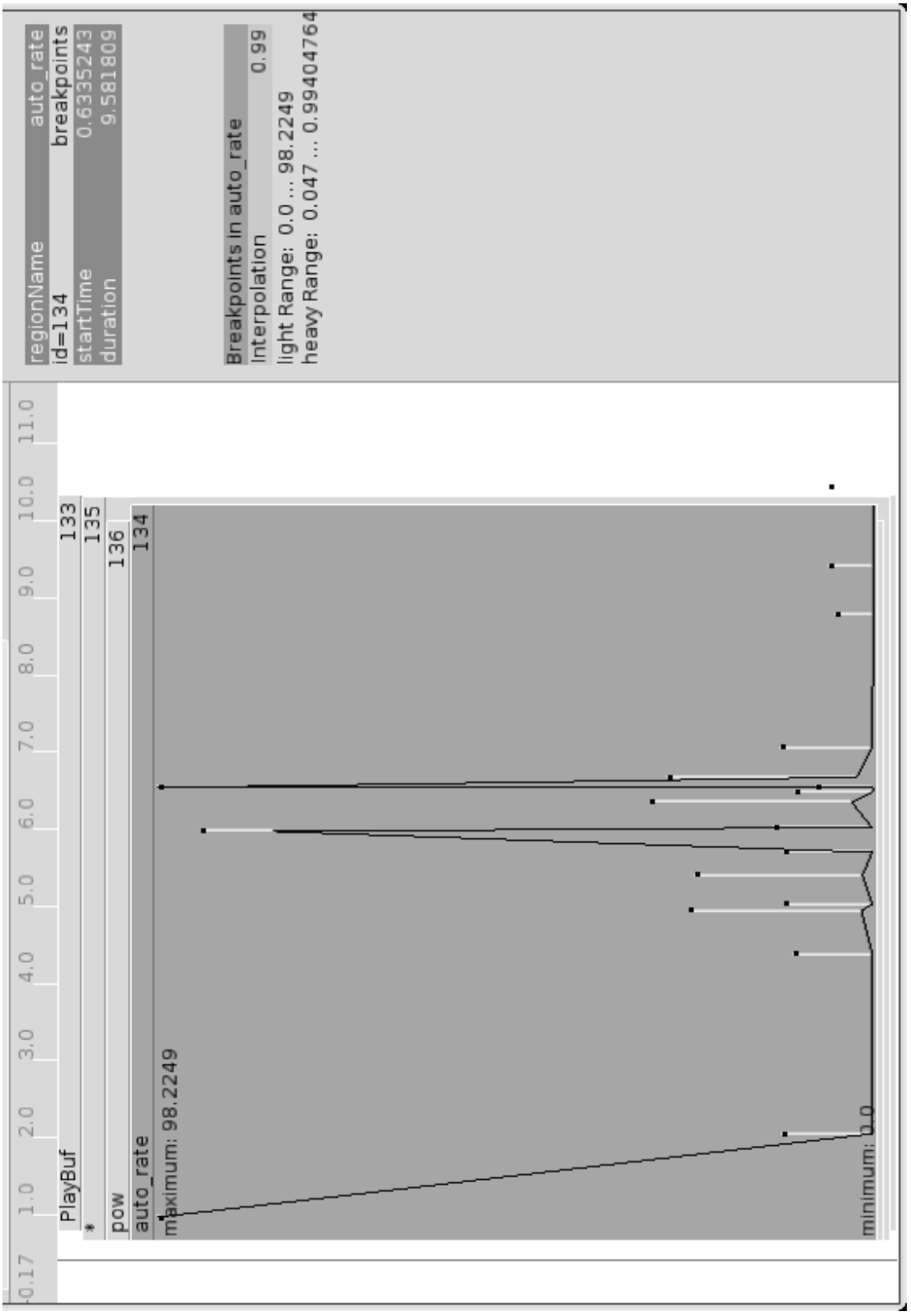

Abbildung A.3: Thema: EnvRegion 134 liefert Werte von 0 bis 1

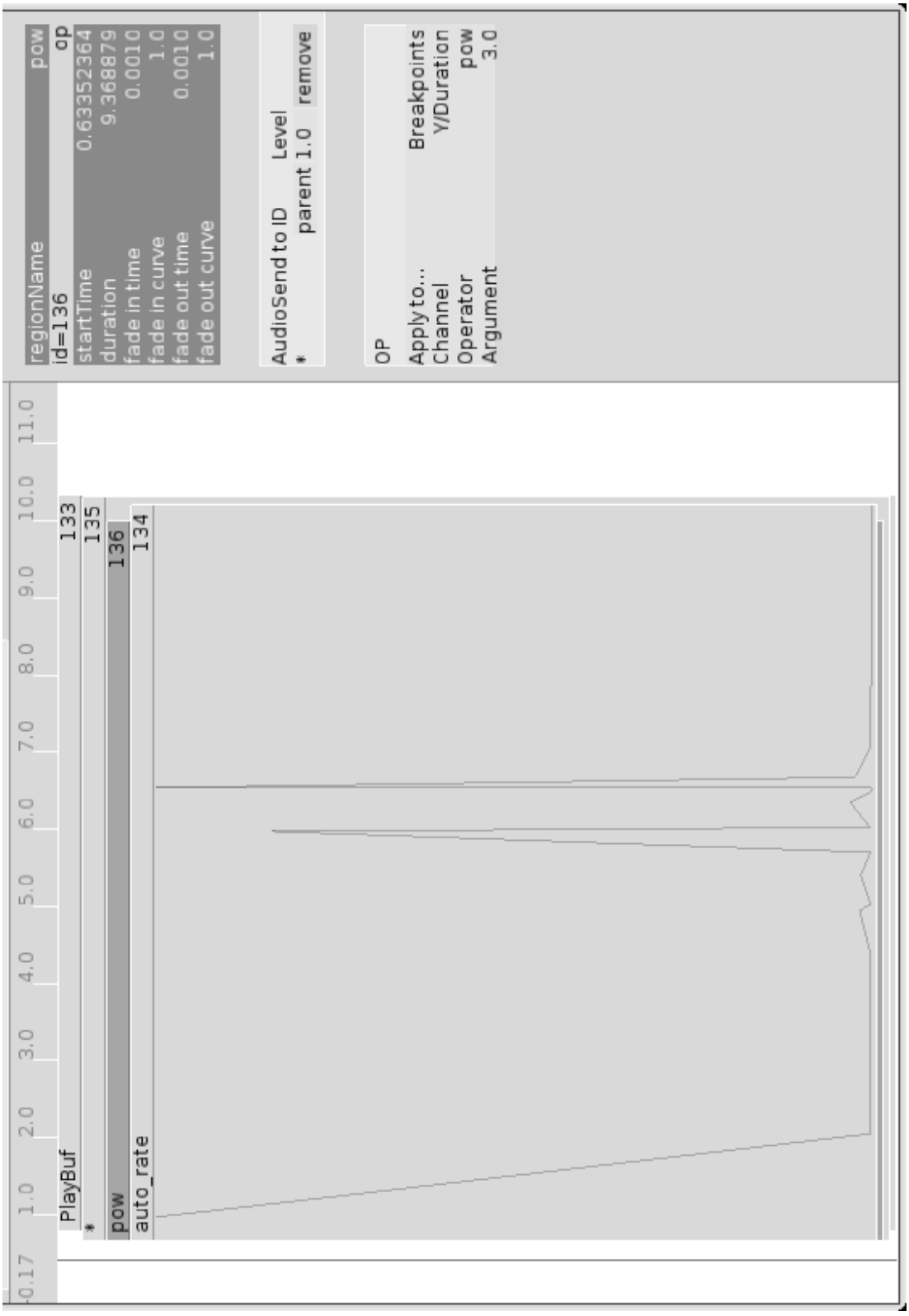

Abbildung A.4: Thema: Der Wertebereich wird durch Potenzieren verformt

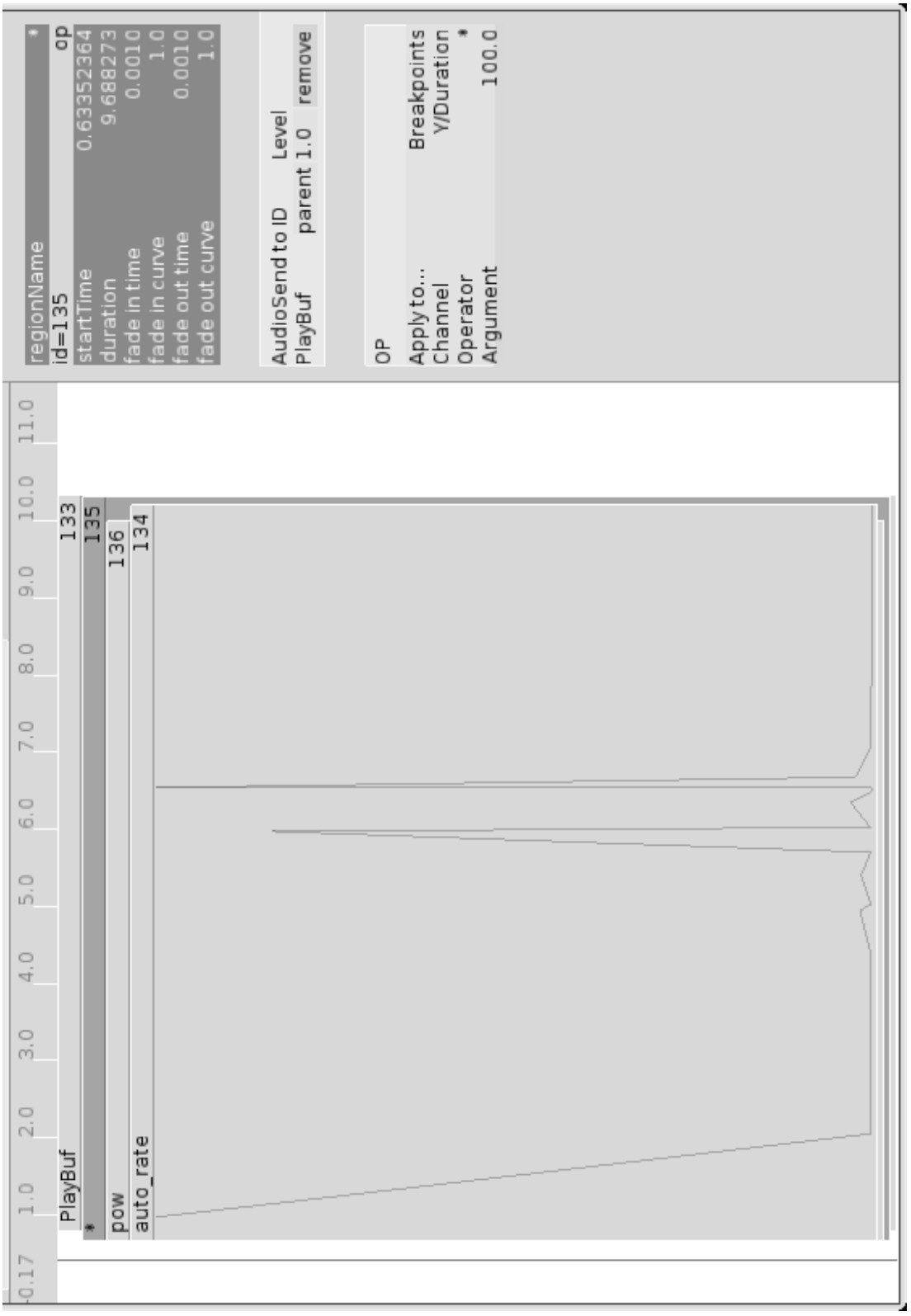

Abbildung A.5: Thema: Anschließend werden die Werte multipliziert

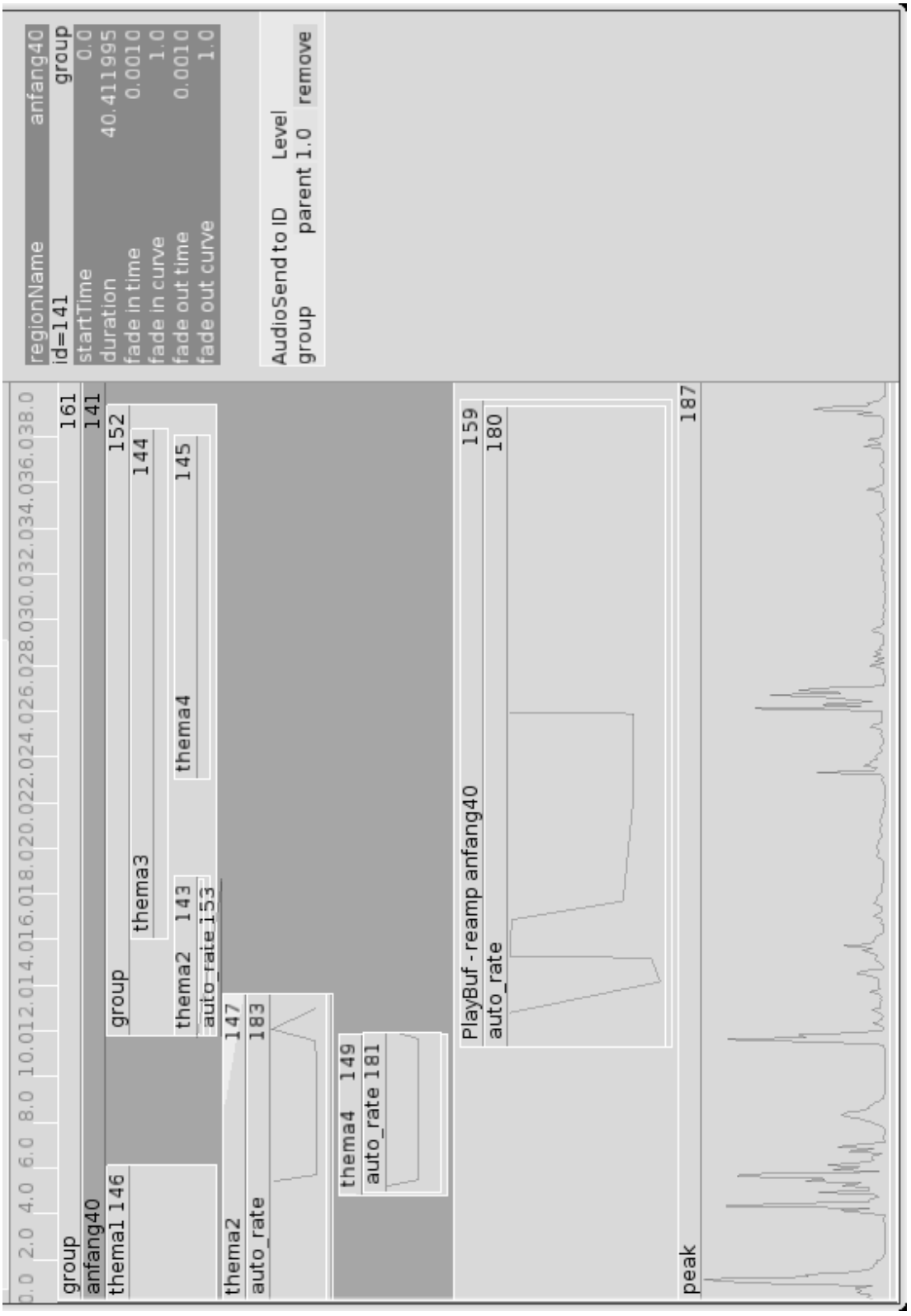

Abbildung A.6: Die ersten 40 Sekunden aus Themenvariationen montiert

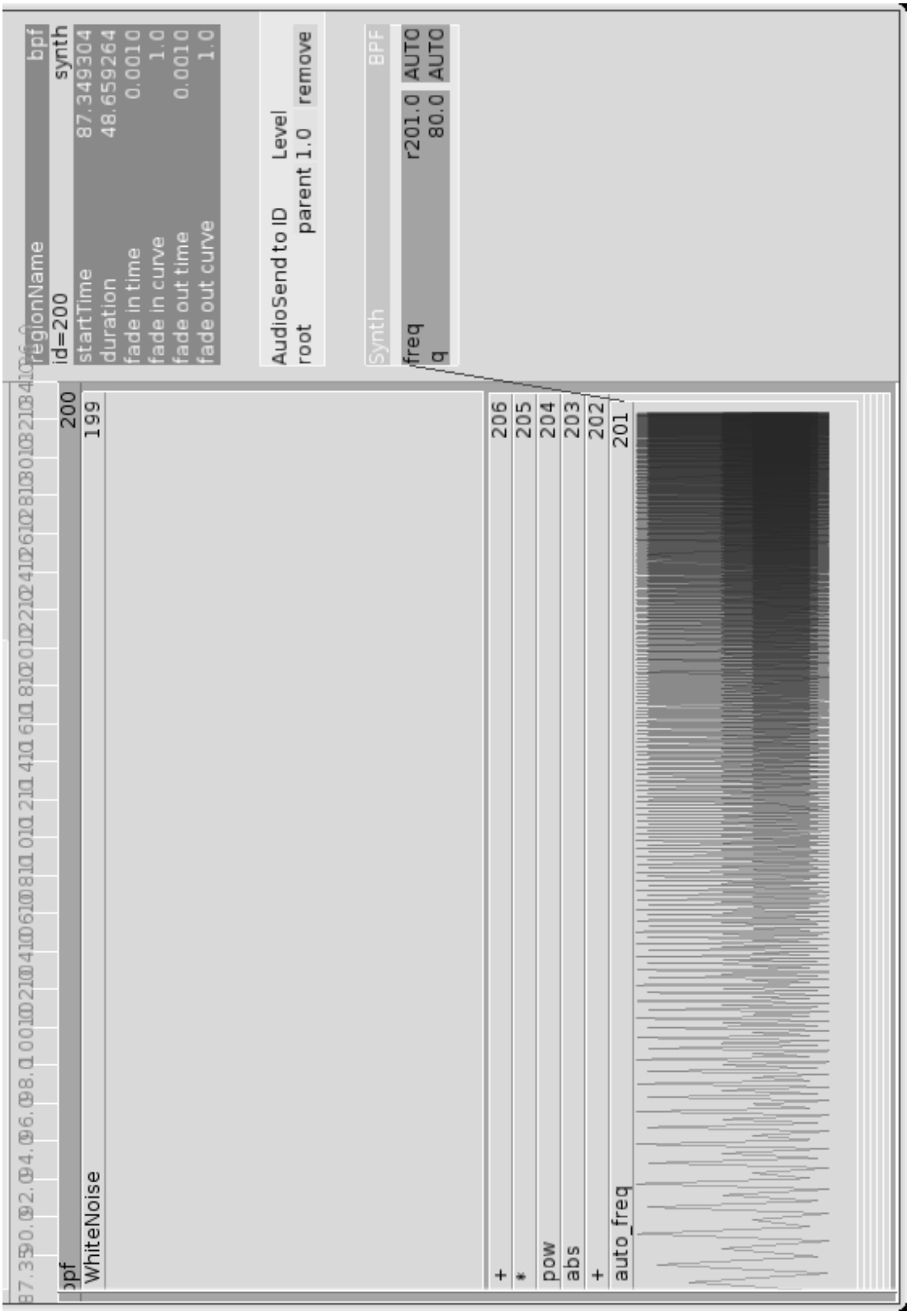

Abbildung A.7: Hénon: 2000 Breakpoints mit 5 OP sind viel für Java

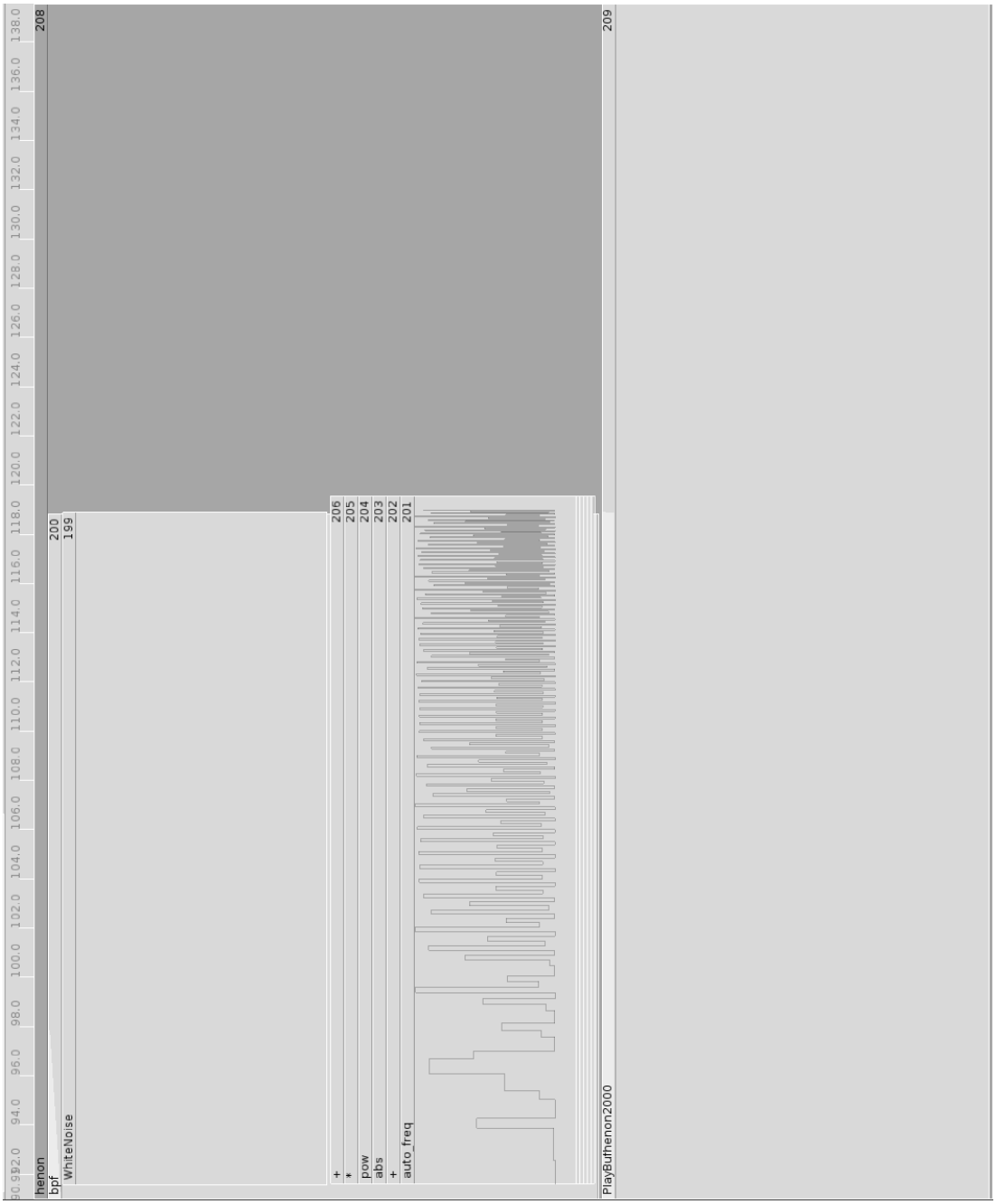

Abbildung A.8: Hénon: Ein Kompromiss - halb Echtzeit, halb gerendert

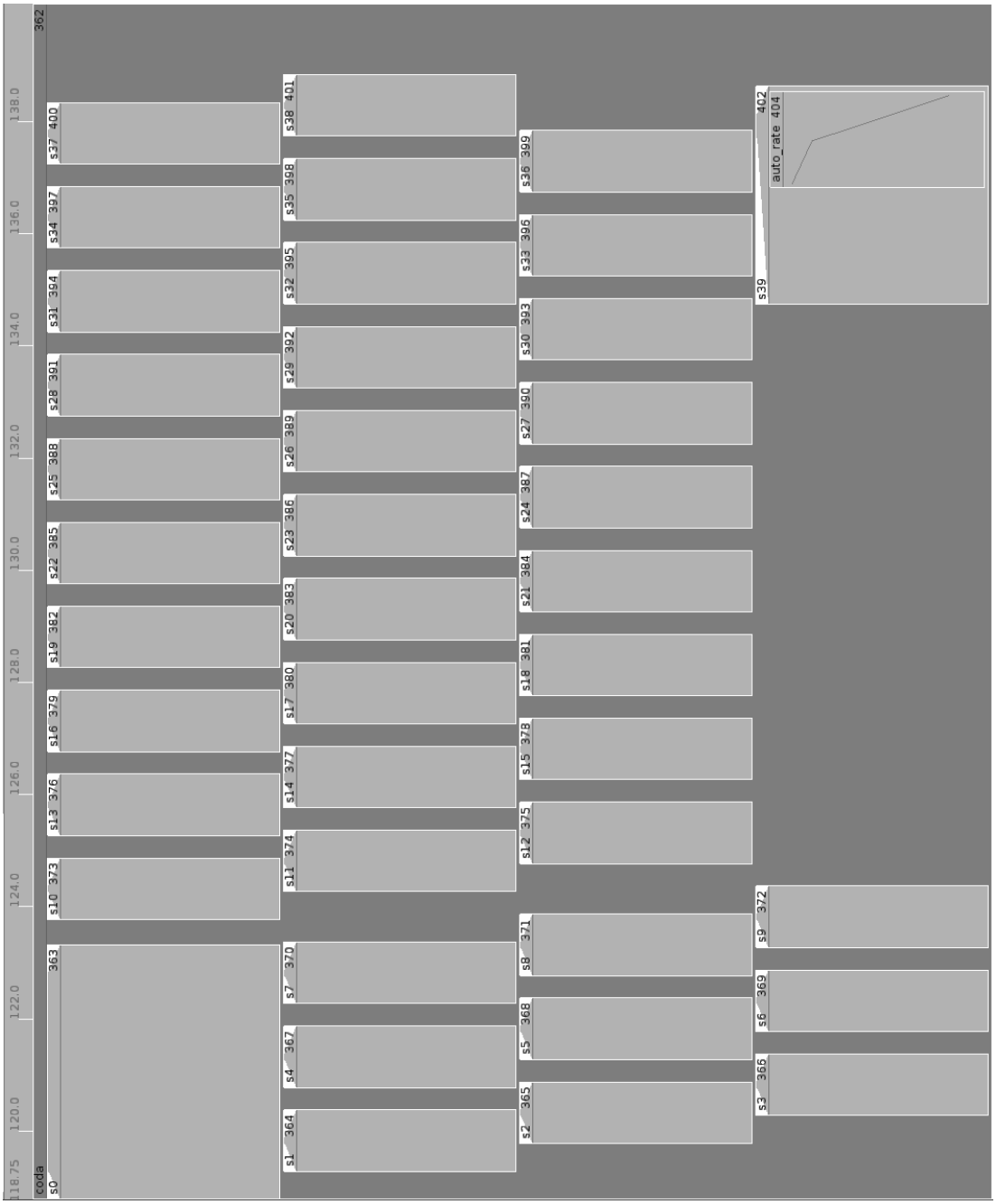

Abbildung A.9: Coda: Partitursynthese aus Klanganalyse

Skript studioluft.sc

```
Y.init

//henon attractor
(
Routine{
var a=1.24, b=0.13, x, y=0,xprev=0, freq, time=e.heavyStartTime_;
2000.do({arg i;
x = 1+y-(xprev**2*a);
y = b*xprev;
//(""+x+y).postln;
xprev=x;
freq=y+0.5;
e.newBreakpoint(time,freq);
time=time+(10/(20+i));
time.postln;
0.001.yield//can't transfer thousands of breakpoints at once
})
}.play
)


//coda: sort anfang40 by peak level
(
{
var javaRegionPeak, javaRegionCoda, codaStart, javaRegionNew, javaFade,
javabreakpoint, breakpoints=[], starttime, duration, bptime, bpvalue;
//get the breakpoints from java
javaRegionPeak = Y.getRegion("peaks vom anfang");
javaRegionPeak.countBreakpoints_.do({arg i;
javabreakpoint = JavaObject.newFrom(javaRegionPeak, \getBreakpoint,i);
bptime  = javabreakpoint.lightStartTime_;
bpvalue = javabreakpoint.lightValueAt_(1);
breakpoints = breakpoints.add([bptime, bpvalue]);
});
//sort them
breakpoints = breakpoints.sort({arg a,b; a[1]<b[1]});
breakpoints.size.postln;
//make new regions in coda
javaRegionCoda = Y.getRegion("coda");
codaStart = javaRegionCoda.lightStartTime_;
breakpoints.do({arg breakpoint, i;
starttime = codaStart+(i*0.5);
duration = 1.1;
javaRegionNew = javaRegionCoda.newChildSynth("s"++i, starttime, duration, "PlayBuf")
javaRegionNew.setParam(0,8);//set buffer
javaRegionNew.setParam(1,breakpoint[0]-1.04);//set offet
javaRegionNew.setParam(2,1);//set rate
javaFade=JavaObject.newFrom(javaRegionNew,\getFade);
javaFade.setInTime(0.5);
javaFade.setOutTime(0.5);
});
}.fork
)
```

Anhang B

Verwendete Software

Tabelle B.1: Software, die im Prototyp Verwendung findet

Name	offizielle Homepage
Processing	www.processing.org
processing.opengl.*	www.processing.org/reference/libraries/opengl
oscP5	www.sojamo.de/libraries/oscP5
netP5	www.sojamo.de/libraries/oscP5
proXML	www.texone.org/proxml
SuperCollider	www.audiosynth.com
SwingOSC	www.sciss.de/swingOSC

Anhang C

Befehlssatz

Tabelle C.1: RegionView: Editierung per Maus

Handgriff	Funktion
einfacher Klick	öffnet die Region im Inspector
Shift+Klick	selektiert den RegionView
Ziehen in der Mitte	verschiebt die Region mit Unterregionen
Ziehen am linken Rand	verschiebt den Startzeitpunkt
Ziehen am rechten Rand	ändert die Dauer
Rechtsklick oder Strg-Klick	öffnet Kontextmenü

Tabelle C.2: Tastaturbefehle in der GUI

Taste	Funktion
Shift-Q	Programm beenden
S	Speichern
Z	Undo
Shift-Z	Redo
R	Record
SPACE	Start/Stop
+	Zoom In
-	Zoom Out
,	Scroll Left
.	Scroll Right
.	Scroll Right
HOME	Zoom Fokussieren auf Root
PAGE UP	Zoom Fokussieren auf Parent
PAGE DOWN	Zoom Fokussieren auf inspizierte Region
BACKSPACE / DELETE	entfernt Breakpoint oder Region (incl. Kinder)

Tabelle C.3: Befehle im Canvas-Kontextmenü

Befehl	implementiert
remove all breakpoints	ja
remove breakpoints in range	ja
import text	nein
export text	nein
export audio	ja
remic	nein
new child Group	ja
new child Synth	ja
new child OP	ja
new child EnvRegion	ja
remove all children	ja
remove	ja
new child Group	ja
new child Synth	ja
new child Env	ja
new parent Group	ja
new parent Synth	ja
new parent OP	ja
add AudioSend	ja
move up	ja
group with selected siblings	ja
remove but keep children	ja
add AudioSend	ja
duplicate	nein
ANALYSIS Peak 100ms	ja
ANALYSIS Peak 1000ms	ja
ANALYSIS Attacks fine	ja
ANALYSIS Attacks rough	ja
SAMPLE EDITOR	nein
locate start	ja
locate end	ja

Anhang D

Quellcode Processing

Dies ist der Processing-Quellcode zum Zeitpunkt der Realisation von „Studioluft“. Die jeweils aktuelle Version ist bei mir auf Anfrage per Email erhältlich.

Y.pde

```
import processing.opengl.*;

boolean debugging=true;

GUI gui;
SynthDatabase synthdatabase;
SessionManager sessionmanager;
Session session;
PFont fontbig, fontsmall;
Transport transport;
Rohr rohr;
int lastdisplayupdatetime;
boolean[] keysdown;
boolean[] keycodesdown;

boolean displayneedsupdate=true;
int clickregistered=0, clickpending=0;
char keypending;
int keycodepending;
PApplet the_applet;

// THE_ESCAPE_KEY=256; //hack to prevent ESC from quitting the app. quit with shift-q

void setup()
{
  size(800,600,OPENGL);
  frameRate(40);
  colorMode(HSB,1,1,1,1);
  fontbig=loadFont("BitstreamVeraSans-Bold-18.vlw");
  fontsmall=loadFont("BitstreamVeraSans-Roman-12.vlw");
  textFont(fontsmall);
  keysdown=new boolean[256];
  keycodesdown=new boolean[257];

  the_applet=this;
  synthdatabase = new SynthDatabase();
  setupSwingOSC();
  sleep(1222);
  rohr= new Rohr();
  synthdatabase.init();
  sleep(1222);
  gui = new GUI();
  gui.init();
  sleep(222);
```

```
  sessionmanager = new SessionManager();
  transport = new Transport();
  session.initForGUI();
}

void draw()
{
   if(frameCount==2)
   sessionmanager.load("/home/stefan/Y/studioluft/studioluft.xml");
  if(clickregistered!=0)
  {
    clickpending=clickregistered;
    clickregistered=0;
  }
  gui.run();
  if(!sessionmanager.blocked)
  {
    transport.run();
    session.run();
  }
  clickpending=0;
}

void mousePressed()
{
  clickregistered=mouseButton;
  keypending=0;
  displayneedsupdate=true;
}

void mouseReleased()
{
  keypending=0;
  displayneedsupdate=true;
}

void mouseDragged()
{
  displayneedsupdate=true;
}

void mouseMoved()
{
  if(mouseX>=0 && mouseY>=0 && mouseX<width && mouseY<height)
    displayneedsupdate=true;
}

void keyPressed()
{
  // println(keyCode+"   "+key);
  if(key==ESC || keyCode==ESC)
  {
    key=CODED;
    keyCode=256;
  }
  keypending=key;
  keycodepending=keyCode;
  if(key==CODED)
  {
    keycodesdown[keyCode]=true;
  }
  else
  {
    keysdown[key]=true;
  }
  displayneedsupdate=true;
}

void keyReleased()
{
  if(key==CODED)
  {
    keycodesdown[keyCode]=false;
  }
  else
  {
    keysdown[key]=false;
  }
}

void fail(String msg)
{
  println("ERROR: "+msg);
  stop();
  exit();
}

void quit()
{
  stop();
  exit();
}

float log10(float x)
{
  return log(x) / log(10);
}

void debug(String msg)
{
  if(debugging)
  {
    println("DEBUG____"+msg);
  }
}

void debug(float msg)
{
```

```
  debug(""+msg);
}

void warn(String msg)
{
  gui.warn();
  println("WARNING____"+msg);
}
boolean mouseOver(float xleft, float ytop, float xright, float ybottom)
{
  return mouseX>xleft && mouseY>ytop && mouseX<xright && mouseY<ybottom;
}

float[] makeFloatArray(float a, float b)
{
  return new float[]{
    a,b                 };
}
FloatInputFieldTarget[]makeFloatInputFieldTargetArray(float a, float b)
{
  return new FloatInputFieldTarget[]{
    new FloatInputFieldTarget(a), new FloatInputFieldTarget(b)                  };
}


void drawCurve(float x1, float y1, float x2, float y2, float xborderleft, float xborderright, float interpolationshape, int nsteps)
{
  if(x2<xborderleft || x1>xborderright)
    return;
  if(interpolationshape==0)
  {
    x1=max(x1, xborderleft);
    x2=min(x2, xborderright);
    line(x1,y1,x2,y1);
    if(x2<xborderright)
      line(x2,y1,x2,y2);
    return;
  }
  float x,y, prevheavyx=x1, prevheavyy=y1;
  if(prevheavyx<xborderleft)
  {
    prevheavyx=xborderleft;
    prevheavyy=y1+(y2-y1)*pow((xborderleft-x1)/(x2-x1), interpolationshape);
  }
  for(float istep=1; istep<=nsteps; istep++)
  {
    x=x1+(x2-x1)*istep/nsteps;
    if(x>xborderleft)
    {
      if(x>xborderright)
      {
        line(prevheavyx,prevheavyy,xborderright,y1+(y2-y1)*pow((xborderright-x1)/(x2-x1), interpolationshape));
        return;
      }
      y=y1+(y2-y1)*pow(istep/nsteps, interpolationshape);
      if(x>xborderleft)
        line(prevheavyx,prevheavyy,x,y);
      prevheavyx=x;
      prevheavyy=y;
    }
  }
}
public class Coord{
  float x,y;
  Coord(float _x, float _y)
  {
    x=_x;
    y=_y;
  }
  Coord copy()
  {
    return new Coord(x,y);
  }
};
void moveFile(String srcPath, String dstPath)
{
  saveBytes(dstPath, loadBytes(srcPath));
}
void sleep(int milliseconds)
{
  try {
    Thread.sleep(milliseconds);
  }
  catch (InterruptedException e) {
  }
}
void  wait_for_synthdatabase()
{
  for(int milliseconds=1; !synthdatabase.ready && milliseconds<3000 ; milliseconds*=2)
    sleep(milliseconds);
  if(!synthdatabase.ready)
    fail("could not init synthdatabase");
      synthdatabase.print();
}
```

SwingOSCPetze.java

```
//processing generates temporary classes with weird names.
//even the applet itself has a temporary identifier.
//SwingOSC can not access these classes directly by name.
//So the processing applet has to smuggle a reference itself
//into this accessible java class
//see http://processing.org/reference/environment/index.html
//-> "programming modes"

import processing.core.*;

public class SwingOSCPetze{
  static PApplet applet;
  public SwingOSCPetze()
  {
  }
  public PApplet getApplet()
  {
    return applet;
  }
};
```

audiosend.pde

```
public class AudioSend{
  MyIDInputFieldTarget destinationID;
  FloatInputFieldTarget level;
  AudioSend(int _destinationID, float _level)
  {
    destinationID= new MyIDInputFieldTarget(_destinationID);
    level=new FloatInputFieldTarget(_level);
  }
  void setSourceRegion(RegionWithChildren region)
  {
    destinationID.sourceregion = region;
  }
  RegionWithChildren getSourceRegion()
  {
    return destinationID.sourceregion;
  }
  public class MyIDInputFieldTarget extends IDInputFieldTarget{
    RegionWithChildren sourceregion;
    MyIDInputFieldTarget(int _intValue)
    {
      super(_intValue);
    }
    void setString(String s)
    {
      int newIntValue;
      try{
        newIntValue=int(Float.valueOf(s).floatValue());
      }
      catch(Exception e){
        newIntValue=0;
      }
      if(newIntValue<-2)
      {
        intValue=-2;
      }
      else if(newIntValue==sourceregion.id)
      {
        warn("can not route a region to itself");
      }
      else if(sourceregion.getRegionByID(newIntValue)==null)
      {
        warn("region does not exist");
        return;
      }
      else if(sourceregion.getRegionByID(newIntValue).isEnvRegion)
      {
        warn("can not route to an EnvRegion");
      }
      else
      {
        intValue=newIntValue;
      }
      gui.inspector.refresh();
    }
  };
  XMLElement getXMLElement()
  {
    XMLElement result=new XMLElement("audiosend");
    result.addAttribute("destinationID",destinationID.getInt());
    result.addAttribute("level", level.getFloat());
    return result;
  }
  ////////////////////////////////// SCRIPTING
```

```
  int getRelativeID()
  {
    return destinationID.getInt();
  }
  int getAbsoluteID()
  {
    if(destinationID.getInt()<0)
      return getSourceRegion().parent.id;
    return destinationID.getInt();
  }
  float getLevel()
  {
    return level.getFloat();
  }
};

public class Fade{
  FloatInputFieldTarget intime, incurve,outtime,outcurve;
    Fade(float _intime, float _incurve, float _outtime, float _outcurve)
  {
    intime= new FloatInputFieldTarget(_intime);
    incurve= new FloatInputFieldTarget(_incurve);
    outtime= new FloatInputFieldTarget(_outtime);
    outcurve= new FloatInputFieldTarget(_outcurve);
  }
  XMLElement getXMLElement()
  {
    XMLElement result=new XMLElement("fade");
    result.addAttribute("intime", intime.getFloat());
    result.addAttribute("incurve", incurve.getFloat());
    result.addAttribute("outtime", outtime.getFloat());
    result.addAttribute("outcurve", outcurve.getFloat());
    return result;
  }
};
```

buffer.pde

```
public class Buffer{
  int bufnum;
  String path;
  Buffer(String _path, int _bufnum)
  {
    path=_path;
    bufnum=_bufnum;
  }
  int getBufnum()
  {
    return bufnum;
  }
  String getPath()
  {
    return path;
  }
  XMLElement getXMLElement()
  {
    XMLElement result=new XMLElement("buffer");
    result.addAttribute("bufnum",bufnum);
    result.addAttribute("path", path);
    return result;
  }
};
```

canvas.pde

```
public class Canvas extends GUIElement{
  int runCount=0;
  float xPositionOfTime0, pixelspersecond;
  RegionView focussedregionview, draggingregionview;
  float dragPointOnHeadFromLeft, dragPointOnHeadFromRight;
  int dragmode;
  Breakpoint draggingbreakpoint=null;
  RegionActionMenu regionActionMenu;
  Canvas(Coord _pos, Coord _size)
  {
    pos=_pos;
    size=_size;
```

```
  regionActionMenu=new RegionActionMenu();
}
void updateAfterUndo()
{
  clearSelections();
  int id=focussedregionview.region.id;
  focusRegionView(session.getRegionByID(id).view);
}
void zoom(float factor)
{
  float centerTime=pos2seconds(mouseX);
  pixelspersecond*=factor;
  xPositionOfTime0= mouseX - centerTime*pixelspersecond;
  displayneedsupdate=true;
}
void updateXY()
{
  session.updateGenerations();
  focussedregionview.updateXYDownTree(pos.y,size.y);
  displayneedsupdate=true;
}
void focusRegionView(RegionView rv)
{
  gui.inspector.updateViews(rv.region);
  focussedregionview=rv;
  pixelspersecond=size.x/max(0.1, focussedregionview.region.heavyDuration());
  xPositionOfTime0=pos.x-focussedregionview.region.heavyStartTime()*pixelspersecond;
  displayneedsupdate=true;
  updateXY();
}
void focusInspected()
{
  if(gui.inspector.region==null)
  {
    println("no Region selected");
    return;
  }
  focusRegionView(gui.inspector.region.view);
}
void focusSession()
{
  focusRegionView(session.rootregion.view);
}
void focusParent()
{
  if(focussedregionview.region.parent!=null)
    focusRegionView(focussedregionview.region.parent.view);
}
float seconds2pos(float t)
{
  return xPositionOfTime0 + t * pixelspersecond;
}
float pos2seconds(float x)
{
  return (x-xPositionOfTime0) / pixelspersecond;
}
void run()
{
  runCount++;
  switch(keypending)
  {
  case '+':
    zoom(1.25);
    focussedregionview.updateXYDownTree(pos.y,size.y);
    keypending=0;
    break;
  case '-':
    zoom(.8);
    focussedregionview.updateXYDownTree(pos.y,size.y);
    keypending=0;
    break;
  case ',':
    xPositionOfTime0+=100;
    focussedregionview.updateXYDownTree(pos.y,size.y);
    keypending=0;
    break;
  case '.':
    xPositionOfTime0-=100;
    focussedregionview.updateXYDownTree(pos.y,size.y);
    keypending=0;
    break;
  }
  switch(keycodepending)
  {
  case 33://page up
    focusParent();
    keypending=0;
    keycodepending=0;
    break;
  case 34://page down
    focusInspected();
    keypending=0;
    keycodepending=0;
    break;
  case 36://home
    focusSession();
    keypending=0;
    keycodepending=0;
    break;
  }
  //    Coord rootpos=new Coord(xPositionOfTime0+focussedregion.heavyStartTime()*pixelspersecond, pos.y);
  //    Coord rootsize=new Coord(focussedregion.heavyDuration()*pixelspersecond, size.y);
```

```
    //runRegion(focussedregion, rootpos, rootsize, 0);
    if(focussedregionview!=null)
      focussedregionview.run();
    lineVertical(seconds2pos(transport.curtime),   color(0,1,1,1));
    lineVertical(seconds2pos(transport.curtime)+1, color(.5,1,0,1));
    lineVertical(seconds2pos(transport.curtime)+2, color(.5,1,0,.5));
    lineVertical(seconds2pos(transport.curtime)+3, color(.5,1,0,.2));
    //  println("canvas.run done");
    regionActionMenu.run();

    switch(keypending)
    {
    case DELETE:
    case BACKSPACE:
      if(gui.inspector.region!=null && gui.inspector.region.parent!=null)
      {
        RegionWithChildren parent=gui.inspector.region.parent;
        parent.removeChild(gui.inspector.region);
        gui.inspector.region=parent;
        updateXY();
      }
      keypending=0;
      break;
    }
  }
  void lineVertical(float x, color c)
  {
    pushMatrix();
    translate(0,0,1);
    stroke(c);
    line(x, pos.y, x, pos.y+size.y);
    popMatrix();
  }
  void rectVertical(float startx, float endx, color strokecolor, color fillcolor)
  {
    pushMatrix();
    translate(0,0,1);
    stroke(strokecolor);
    fill(fillcolor);
    rect(startx, pos.y, endx-startx, pos.y+size.y);
    popMatrix();
  }
  void clearSelections()
  {
    session.rootregion.view.selected=false;
    session.rootregion.view.unselectAncestors();
  }
};



public class RegionActionMenu implements GUIDropMenuTarget{
  String answer="";
  Region region;
  RegionActionMenu()
  {
  }
  void activate(Region _region, Coord pos)
  {
    region=_region;
    gui.inspector.updateViews(region);
    ArrayList optionlist = new ArrayList();
    if(region.isEnvRegion)
    {
      if(((EnvRegion)region).countBreakpoints()>0)
        optionlist.add("remove all breakpoints");
      if(transport.loopstarttime!=transport.loopendtime)
        optionlist.add("remove breakpoints in range");
      optionlist.add("import text");
      optionlist.add("export text");
    }
    else
    {
      optionlist.add("export audio");
      optionlist.add("remic");
      optionlist.add("new child Group");
      optionlist.add("new child Synth");
      optionlist.add("new child OP");
      optionlist.add("new child Env");
      optionlist.add("remove all children");
    }
    if(region.parent!=null)
      optionlist.add("remove");
    optionlist.add("new parent Group");
    optionlist.add("new parent Synth");
    optionlist.add("new parent OP");
    if(region.view.countSelectedSiblings()>0)
      optionlist.add("group with selected siblings");
    if(region.parent!=null &! region.isEnvRegion)
      optionlist.add("remove but keep children");
    if(region.parent!=null)
      optionlist.add("add AudioSend");
    if(region.parent!=null && region.parent.parent!=null &! region.isEnvRegion)
      optionlist.add("move up");
    optionlist.add("duplicate");
    if(!region.isEnvRegion)
    {
      optionlist.add("ANALYSIS Peak 100ms");
      optionlist.add("ANALYSIS Attacks fine");
```

```
    optionlist.add("ANALYSIS Attacks rough");
  }
  else
  {
    optionlist.add("SAMPLE EDITOR");
  }
  optionlist.add("locate start");
  optionlist.add("locate end");
  String []optionarray = new String[optionlist.size()];
  for(int i=0; i<optionlist.size(); i++)
    optionarray[i]=(String)optionlist.get(i);
  gui.dropMenu.activate(this, optionarray, pos, false);
}
void run()
{
  if(answer.length()==0)
    return;
  debug("answer = '"+answer+"'");
  if(answer.equals("remove"))
  {
    if(region==gui.inspector.region)
    {
      gui.inspector.updateViews(region.parent);
      gui.canvas.focusRegionView(region.parent.view);
    }
    else
      gui.inspector.refresh();
    region.parent.removeChild(region);
    gui.canvas.updateXY();
  }
  else if(answer.equals("new child Group"))
    ((RegionWithChildren)region).newChildGroup("group");
  else if(answer.equals("new child Synth"))
    ((RegionWithChildren)region).newChildSynth("WhiteNoise", "WhiteNoise");
  else if(answer.equals("new child Env"))
    ((RegionWithChildren)region).newChildEnv("env",1);
  else if(answer.equals("new parent Group"))
    region.newParentGroup("group");
  else if(answer.equals("new parent Synth"))
    region.newParentSynth("bpf");
  else if(answer.equals("new parent OP"))
    region.newParentOP("op");
  else if(answer.equals("add AudioSend"))
    ((RegionWithChildren)region).addAudioSend(new AudioSend(-2,0));
  else if(answer.equals("move up"))
    ((RegionWithChildren)region).moveUp();
  else if(answer.equals("group with selected siblings"))
  {
    ArrayList selsib=new ArrayList();
    region.view.collectSelectedSiblings(selsib);
    selsib.remove(region);
    GroupRegion newparent=new GroupRegion(region.parent, "Group over "+region.name()+" and others", session.nextFreeID(), makeFloatInputFieldTargetAr
    newparent.init();
    for(int i=0;i<selsib.size();i++)
    {
      Region r=(Region)selsib.get(i);
      region.parent.removeChild(r);
      newparent.addChild(r);
    }
    region.parent.replaceChild(region,newparent);
    newparent.addChild(region);
    gui.inspector.updateViews(newparent);
    gui.canvas.updateXY();
    gui.canvas.clearSelections();
  }
  else if(answer.equals("remove but keep children"))
  {
    RegionWithChildren regionwithchildren = (RegionWithChildren)region;
    for(int i=regionwithchildren.countChildren()-1;i>=0;i--)
    {
      Region child=regionwithchildren.getChild(i);
      regionwithchildren.removeChild(child);
      region.parent.addChild(child);
    }
    region.parent.removeChild(region);
    gui.inspector.updateViews(region.parent);
    gui.canvas.updateXY();
  }
  else if(answer.equals("remove all breakpoints"))
  {
    ((EnvRegion)region).removeAllBreakpoints();
  }
  else if(answer.equals("remove breakpoints in range"))
  {
    ((EnvRegion)region).removeBreakpointsInTimeRange(transport.loopstarttime, transport.loopendtime);
  }
  else if(answer.equals("remove all children"))
  {
    ((RegionWithChildren)region).removeAllChildren();
    gui.inspector.updateViews(region);
    gui.canvas.updateXY();
  }
  else if(answer.equals("export audio"))
  {
    rohr.send("/export", gui.inspector.region.id);
  }
  else if(answer.equals("ANALYSIS Peak 100ms"))
  {
    rohr.send("/analyse", "peak", gui.inspector.region.id, .1);
```

```
    }
    else if(answer.equals("ANALYSIS Attacks fine"))
    {
      rohr.send("/analyse", "attacks", gui.inspector.region.id, 0, .2, .05);
    }
    else if(answer.equals("ANALYSIS Attacks rough"))
    {
      rohr.send("/analyse", "attacks", gui.inspector.region.id, .9, .2, .08);
    }
    else if(answer.equals("locate start"))
    {
      transport.locate(gui.inspector.region.heavyStartTime());
    }
    else if(answer.equals("locate end"))
    {
      transport.locate(gui.inspector.region.heavyEndTime());
    }
    else if(answer.equals("SAMPLE EDITOR"))
    {
      rohr.send("/sampleeditor", gui.inspector.region.id);
    }
    else
      warn("region action '"+answer+"' not yet implemented");
    gui.canvas.updateXY();
    displayneedsupdate=true;
    answer="";
    session.markDirty();
  }
  private void initNewParent(RegionWithChildren newparent)
  {
    newparent.init();
    newparent.addChild(region);
    if(newparent.parent==null)
    {
      session.rootregion.destroy();
      session.rootregion=newparent;
      session.updateGenerations();
      gui.canvas.focusRegionView(newparent.view);
    }
    else
    {
      RegionWithChildren oldparent = newparent.parent;
      oldparent.replaceChild(region,newparent);
      gui.canvas.updateXY();
    }
  }
  void setAnswerIndex(int index)
  {
  }
  void setAnswerString(String answerString)
  {
    answer=answerString;
  }
};
```

gui.pde

```
public class GUI{
  Canvas canvas;
  Coord ctpos, ctsize;
  Inspector inspector;
  TimeLineView timeline;
  MasterView masterview;
  float warnlevel;
  GUITextInputField textInputField;
  GUIDropMenu dropMenu;
  color backgroundcolor;
  Coord margin;
  GUI()
  {
    margin=new Coord(10,10);
  }
  void init()
  {
    int inspectorposx=width-230;
    masterview = new MasterView(new Coord(0,0), new Coord(inspectorposx, 50));
    timeline = new TimeLineView(new Coord(0,masterview.size.y), new Coord(inspectorposx, 30));
    canvas = new Canvas(new Coord(0,timeline.pos.y+timeline.size.y), new Coord(inspectorposx,height-timeline.pos.y-timeline.size.y-1));
    inspector = new Inspector(new Coord(inspectorposx,0), new Coord(width-inspectorposx,height-1));
    textInputField = new GUITextInputField();
    dropMenu = new GUIDropMenu();
  }
  void run()
  {
    if(warnlevel>0)////////////////////  W A R N I N G S
    {
      fill(color(0,warnlevel,1));
      pushMatrix();
      translate(0,0,20);
      rect(0,0,width,height);
```

```
      popMatrix();
      warnlevel-=.2;
      displayneedsupdate=true;
      return;
    }
    if(textInputField.active)////////////////////// T E X T F I E L D
    {
      textInputField.run();
      return;
    }
    if(dropMenu.active)////////////////////// D R O P   M E N U
    {
      dropMenu.run();
      return;
    }
    ////////////////////////////////////// D I S P L A Y U P D A T E
    if(gui.masterview.snowbutton.active)
      displayneedsupdate=true;
    if(!displayneedsupdate && clickpending==0 && keypending==0 && lastdisplayupdatetime>millis()-400)
      return;
    lastdisplayupdatetime=millis();
    displayneedsupdate=false;
    if(!sessionmanager.blocked)
      background(0);

    switch(keypending)//////////////////////  KEYBOARD SHORTCUTS WHICH DON'T NEED THE SESSION
    {
    case 'Q':
      session.save();
      quit();
      break;
    }
    backgroundcolor = inspector.recordbutton.active ? (gui.inspector.region.isEnvRegion ? color(.15,1,.8) :color(0,1,.8)) :
    color(.7,1,.3);
    masterview.run();///////////////// M A S T E R V I E W
    if(sessionmanager.blocked)
    {
      noStroke();
      fill(0,1,1,.5);
      rect(width*3,height*.4,width*4,height*.2);
      fill(0,1,1,1);
      String msg="S E S S I O N   B L O C K E D";
      text(msg, (width-textWidth(msg))/2,height/2);
      return;
    }
    inspector.run();///////////////// I N S P E C T O R
    timeline.run();///////////////// T I M E L I N E
    canvas.run(); ///////////////// C A N V A S
    timeline.display();//must be after Regions in Canvas because of transparency
      switch(keypending)////////////////////// KEYBOARD SHORTCUTS WHICH NEED THE SESSION
    {
    case 's':
      session.saveForced();
      keypending=0;
      break;
    case 'z':
      sessionmanager.undo();
      keypending=0;
      break;
    case 'Z':
      sessionmanager.redo();
      keypending=0;
      break;
    case 'l':
      sessionmanager.reload();
      keypending=0;
      break;
    case 'r':
      inspector.recordbutton.active = !inspector.recordbutton.active;
      keypending=0;
      break;
    case ' ':
      transport.togglePlay();
      keypending=0;
      break;
    }
    //  println("gui.run done");
  }
  void warn()
  {
    warnlevel=1;
  }
};


interface GUIDropMenuTarget{
  void setAnswerIndex(int index);
  void setAnswerString(String answerString);
};

abstract public class GUITextInputFieldTarget{
  abstract void setString(String s);
  abstract String getString();
  XMLElement getXMLElement(String name)
  {
    XMLElement result = new XMLElement(name);
    result.addAttribute("value",getString());
    return result;
  }
};
```

```
public class FloatInputFieldTarget extends GUITextInputFieldTarget{
  private float floatValue;
  FloatInputFieldTarget(float _floatValue)
  {
    floatValue=_floatValue;
  }
  void setString(String s)
  {
    try{
      floatValue=Float.valueOf(s).floatValue();
    }
    catch(Exception e){
      floatValue=0;
    }
    session.markDirty();
  }
  float getFloat()
  {
    return floatValue;
  }
  void setFloat(float x)
  {
    floatValue=x;
  }
  String getString()
  {
    return ""+floatValue;
  }
};

public class IDInputFieldTarget extends GUITextInputFieldTarget{
  protected int intValue;
  IDInputFieldTarget(int _intValue)
  {
    intValue=_intValue;
  }
  void setString(String s)
  {
    try{
      intValue=int(Float.valueOf(s).floatValue());
    }
    catch(Exception e){
      intValue=0;
    }
    session.markDirty();
  }
  int getInt()
  {
    return intValue;
  }
  void setInt(int x)
  {
    intValue=x;
  }
  String getString()
  {
    if(intValue<0)
    {
      if(intValue==-1)
        return "parent";
      intValue=-2;
      return "root";
    }
    return ""+intValue;
  }
};

public class FloatOrRegionInputFieldTarget extends GUITextInputFieldTarget{
  float floatValue;
  boolean isrouting;
  FloatOrRegionInputFieldTarget(float _floatValue, boolean _isrouting)
  {
    floatValue=_floatValue;
    isrouting=_isrouting;
  }
  void setString(String s)
  {
    if(s==null || s.length()==0)
      return;
    float inputAsFloat;
    if(s.charAt(0)=='r')
    {
      if(s.length()<2)
        return;
      try{
        inputAsFloat=Float.valueOf(s.substring(1,s.length())).floatValue();
      }
      catch(Exception e){
        warn("you entered crap in a FloatOrRegionInputFieldTarget: "+s);
        return;
      }
      int id=int(inputAsFloat+.5);
      if(session.getRegionByID(id)==null)
      {
        warn("region ID "+id+" does not exist");
        return;
      }
      floatValue=id;
      isrouting=true;
    }
    else try{
      isrouting=false;
      floatValue=Float.valueOf(s).floatValue();
    }
```

```
    catch(Exception e){
      floatValue=0;
    }
    session.markDirty();
  }
  float getFloat()
  {
    if(isrouting)
      return 0;
    return floatValue;
  }
  void setFloat(float x)
  {
    isrouting=false;
    floatValue=x;
  }
  int getID()
  {
    if(isrouting)
      return int(floatValue+.5);
    fail("programming error: FloatOrRegionInputFieldTarget.getID called when isrouting==false");
    return 0;//never reached
  }
  void setID(int id)
  {
    isrouting=true;
    floatValue=id;
  }
  String getString()
  {
    if(isrouting)
      return "r"+floatValue;
    return ""+floatValue;
  }
};

public class StringInputFieldTarget extends GUITextInputFieldTarget{
  protected String stringValue;
  StringInputFieldTarget(String _stringValue)
  {
    stringValue=_stringValue;
  }
  void setString(String s)
  {
    stringValue=s;
    session.markDirty();
  }
  String getString()
  {
    return stringValue;
  }
};


////////////////////////////////////////    G U I   E L E M E N T       ////////////////
abstract public class GUIElement{
  Coord pos, size;
  abstract void run();
};


public class GUILabel extends GUIElement{
  color textColor;
  String title;
  GUILabel(String _title, color _textColor, Coord _pos, Coord _size)
  {
    title=_title;
    textColor=_textColor;
    pos=_pos;
    size=_size;
  }
  final void run()
  {
    display();
  }
  void display()
  {
    fill(textColor);
    text(title, pos.x, pos.y+size.y-4);
  }
};


public class GUIButton extends GUIElement{
  boolean wasclicked=false;
  color foreColor, backColorNormal, backColorMouseover;
  String title;
  GUIButton(String _title, color _foreColor, color _backColorNormal, color _backColorMouseover, Coord _pos, Coord _size)
  {
    title=_title;
    foreColor=_foreColor;
    backColorNormal=_backColorNormal;
    backColorMouseover=_backColorMouseover;
    pos=_pos;
    size=_size;
  }
  final void run()
  {
    if(mouseOver() && clickpending==LEFT)
    {
      clickpending=0;
      wasclicked=true;
      click();
```

```
    }
    display();
  }
  void click()
  {
  }
  void display()
  {
    noStroke();
    fill(mouseOver() ? backColorMouseover : backColorNormal);
    rect(pos.x,pos.y, size.x,size.y);
    fill(foreColor);
    text(title, pos.x+1, pos.y+size.y-4);
  }
  final boolean mouseOver()
  {
    return mouseX>=pos.x && mouseX<pos.x+size.x && mouseY>=pos.y && mouseY<pos.y+size.y;
  }
};
public class RecordButton extends GUIButton{
  boolean active;
  RecordButton(Coord _pos, Coord _size)
  {
    super("RECORD", 0,0,0, _pos, _size);
  }
  void click()
  {
    active = !active;
    if(active)
      clickON();
    else
      clickOFF();
  }
  void clickON()
  {
  }
  void clickOFF()
  {
  }
  void display()
  {
    boolean midimode = gui.inspector.region!=null && gui.inspector.region.isEnvRegion;
    noStroke();
    float colorhue = midimode ? .15 : 0;
    fill(mouseOver() ? color(colorhue,1,1) : color(colorhue,1,.4));
    rect(pos.x,pos.y, size.x,size.y);
    fill(foreColor);
    text(midimode ? "REC MIDI":"REC AUD", pos.x+1, pos.y+size.y-4);
  }
};
public class GUIStringEntryButton extends GUIButton{
  StringInputFieldTarget target;
  GUIStringEntryButton(StringInputFieldTarget _target, String _title, color _foreColor, color _backColorNormal, color _backColorMouseover, Coord _pos,
  {
    super(_title, _foreColor, _backColorNormal, _backColorMouseover, _pos, _size);
    target=_target;
  }
  void click()
  {
    gui.textInputField.activate(target, target.getString(), false, false, true, pos.copy());
  }
  final void display()
  {
    noStroke();
    fill(mouseOver() ? backColorMouseover : backColorNormal);
    rect(pos.x,pos.y, size.x,size.y);
    fill(foreColor);
    text(title, pos.x+1, pos.y+size.y-4);
    text(target.getString(), pos.x+size.x-textWidth(target.getString())-2, pos.y+size.y-4);
  }
};
public class GUIFloatEntryButton extends GUIButton{
  FloatInputFieldTarget target;
  GUIFloatEntryButton(FloatInputFieldTarget _target, String _title, color _foreColor, color _backColorNormal, color _backColorMouseover, Coord _pos, Co
  {
    super(_title, _foreColor, _backColorNormal, _backColorMouseover, _pos, _size);
    target=_target;
  }
  final void display()
  {
    noStroke();
    fill(mouseOver() ? backColorMouseover : backColorNormal);
    rect(pos.x,pos.y, size.x,size.y);
    fill(foreColor);
    if(title.length()==0)
      text(target.getString(), pos.x, pos.y+size.y-4);
    else
    {
      text(title, pos.x+1, pos.y+size.y-4);
      text(target.getString(), pos.x+size.x-textWidth(target.getString())-2, pos.y+size.y-4);
    }
  }
  void click()
  {
    //    gui.textInputField.activate(target, ""+target.getFloat(), true, false, pos.copy());
    gui.textInputField.activate(target, "", true, false, true, pos.copy());
  }
};
public class GUIIDEntryButton extends GUIButton{
  IDInputFieldTarget target;
  GUIIDEntryButton(IDInputFieldTarget _target, String _title, color _foreColor, color _backColorNormal, color _backColorMouseover, Coord _pos, Coord _s
```

```
  {
    super(_title, _foreColor, _backColorNormal, _backColorMouseover, _pos, _size);
    target=_target;
  }
  final void display()
  {
    noStroke();
    fill(mouseOver() ? backColorMouseover : backColorNormal);
    rect(pos.x,pos.y, size.x,size.y);
    fill(foreColor);
    if(title.length()==0)
      text(target.getString(), pos.x, pos.y+size.y-4);
    else
    {
      text(title, pos.x+1, pos.y+size.y-4);
      text(target.getString(), pos.x+size.x-textWidth(target.getString())-2, pos.y+size.y-4);
    }
  }
  void click()
  {
    //    gui.textInputField.activate(target, ""+target.getFloat(), true, false, pos.copy());
    gui.textInputField.activate(target, "", true, false, false, pos.copy());
  }
};
public class GUIFloatOrRegionEntryButton extends GUIButton{
  FloatOrRegionInputFieldTarget target;
  GUIFloatOrRegionEntryButton(FloatOrRegionInputFieldTarget _target, String _title, color _foreColor, color _backColorNormal, color _backColorMouseover
  {
    super(_title, _foreColor, _backColorNormal, _backColorMouseover, _pos, _size);
    target=_target;
  }
  final void display()
  {
    noStroke();
    fill(mouseOver() ? backColorMouseover : backColorNormal);
    rect(pos.x,pos.y, size.x,size.y);
    fill(foreColor);
    if(title.length()==0)
      text(""+target.getFloat(), pos.x, pos.y+size.y-4);
    else
    {
      text(title, pos.x+1, pos.y+size.y-4);
      text(""+target.getString(), pos.x+size.x-textWidth(""+target.getString())-2, pos.y+size.y-4);
    }
  }
  void click()
  {
    gui.textInputField.activate(target, "", true, true, true, pos.copy());
  }
};


public class GUITextInputField extends GUIElement{
  GUITextInputFieldTarget target;//e.g. a Param
  boolean active, dirty;
  String content;
  boolean numbermode, allowarrow, allowperiod;
  GUITextInputField()
  {
  }
  void activate(GUITextInputFieldTarget _target, String _content, boolean _numbermode, boolean _allowarrow, boolean _allowperiod, Coord _pos)
  {
    active=true;
    dirty=false;
    //    println("GUITextInputField.activate");
    target=_target;
    numbermode=_numbermode;
    allowarrow=_allowarrow;
    allowperiod=_allowperiod;
    content = _content;
    pos=_pos;
    size=new Coord(max(40,textWidth(content)+8), 18);
  }
  void deactivate()
  {
    active=false;
  }
  void saveAndDeactivate()
  {
    target.setString(content);
    active=false;
  }
  void run()
  {
    if(!active)
      return;
    if(clickpending==LEFT)
    {
      clickpending=0;
      if(dirty)
        saveAndDeactivate();
      deactivate();
      return;
    }
    if(keypending!=0)
      handleKeys();
    pushMatrix();
    translate(0,0,5);
    fill(.3,.3,.3);
    stroke(1);
    rect(pos.x-2, pos.y, size.x+2, size.y);
    stroke(1);
```

```
    fill(1);
    textFont(fontsmall);
    text(""+content, pos.x, pos.y+15);
    noStroke();
    fill(0,0,0,.04);
    rect(pos.x,pos.y,size.x+10,size.y+10);
    popMatrix();
  }
  void handleKeys()
  {
    switch(keypending)
    {
    case ENTER:
    case RETURN:
      saveAndDeactivate();
      keypending=0;
      break;
    case BACKSPACE:
    case DELETE:
      if(numbermode)
        content="";
      else if(content.length()>0)
        content=content.substring(0,content.length()-1);
      keypending=0;
      break;
    }
    if(keypending==CODED)
    {
      switch(keycodepending)
      {
      case 256://ESC replacement
        deactivate();
        keypending=0;
        break;
      }
      keypending=0;
    }
    if(keypending!=0)//still
    {
      if(keypending!=CODED || keycodepending==SHIFT || keycodepending==CONTROL || keycodepending==ALT)
      {
        if(!numbermode || (keypending>='0' && keypending<='9') || keypending=='-' || (allowperiod && keypending=='.') || (allowarrow && content.length(
          content+=char(keypending);
        if(textWidth(content)>size.x)
        {
          int pixelstoomuch=int(textWidth(content)-size.x);
          size.x+=pixelstoomuch;
          if(pos.x>gui.margin.x)
            pos.x-=pixelstoomuch;
        }
        keypending=0;
      }
    }
  }
};


public class GUIDropMenu extends GUIElement{
  GUIDropMenuTarget target;
  String []options;
  GUIButton[]buttons;
  boolean active;
  GUIDropMenu()
  {
  }
  void activate(GUIDropMenuTarget _target, String []_options, Coord _pos, boolean rightalign)
  {
    active=true;
    target=_target;
    options=_options;
    pos=_pos;
    buttons=new GUIButton[options.length];
    int buttonheight=gui.inspector.lineheight;
    size=new Coord(20,options.length*buttonheight);
    for(int i=0;i<options.length;i++)
    {
      if(textWidth(options[i])>size.x)
        size.x = textWidth(options[i]);
    }
    size.x+=4;
    if(rightalign)
      pos.x=max(pos.x-size.x, 0);
    pos.x=constrain(pos.x, 10, width-size.x);
    pos.y=constrain(pos.y, 10, height-size.y);
    for(int i=0;i<options.length;i++)
      buttons[i] = new GUIButton(options[i], color(0,0,1), color(0,0,.2), color(0,0,.5), new Coord(pos.x,pos.y+buttonheight*i), new Coord(size.x,button
  }
  void deactivate()
  {
    active=false;
  }
  void run()
  {
    if(!active)
      return;
    if(keypending==CODED)
      switch(keycodepending)
      {
      case 256://ESC replacement
        deactivate();
        keypending=0;
        break;
```

```
      }
    pushMatrix();
    translate(0,0,2);
    for(int ibutton=0;ibutton<buttons.length;ibutton++)
    {
      buttons[ibutton].run();
      if(buttons[ibutton].wasclicked)
      {
        target.setAnswerIndex(ibutton);
        target.setAnswerString(options[ibutton]);
        deactivate();
        return;
      }
    }
    noStroke();
    fill(0,0,0,.01);
    rect(pos.x,pos.y,size.x+10,size.y+10);
    popMatrix();
    if(clickpending!=0)
    {
      clickpending=0;
      deactivate();
      return;
    }
  }
};


public class GUIDropMenuButton extends GUIButton implements GUIDropMenuTarget{ //click opens a dropmenu
  int answer=-1;//index of user choice
  String answerString;
  int selection;
  String[]options;
  boolean rightalign;
  GUIDropMenuButton(String _title, String[]_options, int defaultoption, color _foreColor, color _backColorNormal, color _backColorMouseover, Coord _pos
  {
    super(_title, _foreColor, _backColorNormal, _backColorMouseover, _pos, _size);
    selection=defaultoption;
    options=_options;
    rightalign=_rightalign;
  }
  void click()
  {
    gui.dropMenu.activate(this, options, rightalign ? new Coord(pos.x+size.x, pos.y): pos.copy(), rightalign);
  }
  final void display()
  {
    noStroke();
    fill(mouseOver() ? backColorMouseover : backColorNormal);
    rect(pos.x,pos.y, size.x,size.y);
    fill(foreColor);
    text(title, pos.x, pos.y+size.y-4);
    text(options[selection], pos.x+size.x-textWidth(options[selection])-2, pos.y+size.y-4);
  }
  void setAnswerIndex(int index)
  {
    answer=index;
    selection=answer;
  }
  void setAnswerString(String _answerString)
  {
    answerString=_answerString;
  }
};
```

inspector.pde

```
public class Inspector extends GUIElement{
  ArrayList headerguielements, synthviews, opviews, envregionviews, audiosendviews;
  GUILabel audiosendlabel_id, audiosendlabel_level;
  int linewidth, lineheight = 15;
  Region region=null, selectedenvregion=null;
  RecordButton recordbutton;
  Inspector(Coord _pos, Coord _size)
  {
    pos=_pos;
    size=_size;
    linewidth=int(width-gui.margin.x*2-pos.x);
    recordbutton=new RecordButton(new Coord(pos.x+size.x-55-gui.margin.x,pos.y+gui.margin.y), new Coord(55,20));
  }
  void refresh()
  {
    updateViews(region);
  }
  void updateViews(Region _region)
  {
    if(_region==null)
      return;
    region=_region;
    headerguielements = new ArrayList();
```

```
  synthviews = new ArrayList();
  opviews = new ArrayList();
  envregionviews = new ArrayList();
  audiosendviews = new ArrayList();
  int x = int(pos.x+gui.margin.x);
  //    println(region.typename);
  //HEADER
  headerguielements.add(new GUIStringEntryButton(region.namefieldtarget, "regionName", color(0,0,0), color(.6,.5,1,.5), color(.6,.5,1,1), new Coord(x
  headerguielements.add(new GUILabel("id="+region.id, color(0,0,1), new Coord(x,0), new Coord(linewidth,lineheight)));
  headerguielements.add(new GUILabel(""+region.typename, color(0,0,1), new Coord(x+linewidth-textWidth(""+region.typename),0), new Coord(linewidth,li
  headerguielements.add(new GUIFloatEntryButton(region.heavyValueAt(0), "startTime", color(0,0,0), color(.6,.5,1,.5), color(.6,.5,1,1), new Coord(x,0
  headerguielements.add(new GUIFloatEntryButton(region.heavyValueAt(1), "duration", color(0,0,0), color(.6,.5,1,.5), color(.6,.5,1,1), new Coord(x,0)
  //FADES
  if(!region.isEnvRegion)
  {
    headerguielements.add(new GUIFloatEntryButton(((RegionWithChildren)region).fade.intime, "fade in time", color(0,0,0), color(.5,.5,1,.5), color(.5
    headerguielements.add(new GUIFloatEntryButton(((RegionWithChildren)region).fade.incurve, "fade in curve", color(0,0,0), color(.5,.5,1,.5), color(
    headerguielements.add(new GUIFloatEntryButton(((RegionWithChildren)region).fade.outtime, "fade out time", color(0,0,0), color(.5,.5,1,.5), color(
    headerguielements.add(new GUIFloatEntryButton(((RegionWithChildren)region).fade.outcurve, "fade out curve", color(0,0,0), color(.5,.5,1,.5), colo
  }
  //AUDIOSENDS
  audiosendlabel_id   = new GUILabel("AudioSend to ID", color(1), new Coord(x, 0), new Coord(0, lineheight));
  audiosendlabel_level= new GUILabel("Level", color(1), new Coord(x+linewidth-83, 0), new Coord(0, lineheight));
  if(!region.isEnvRegion)
    for(int isend=0; isend<((RegionWithChildren)region).countAudioSends(); isend++)
    {
      InspectorAudioSendView av = new InspectorAudioSendView(((RegionWithChildren)region).getAudioSend(isend), x,linewidth);
      audiosendviews.add(av);
    }
  //OP
  ArrayList ops = new ArrayList();
  if(region.isOPRegion)
    ops.add(((OPRegion)region).getOP());
  for(int iop=0; iop<ops.size(); iop++)
  {
    InspectorOPView ov=new InspectorOPView((OP)ops.get(iop), x, linewidth);
    opviews.add(ov);
  }
  //SYNTHS
  ArrayList synths = new ArrayList();
  if(region.isSynthRegion)
    //  ((SynthRegion)region).collectSynthsUpTree(synths);
    synths.add(((SynthRegion)region).getSynth());
  for(int isynth=synths.size()-1; isynth>=0; isynth--)
  {
    InspectorSynthView sv=new InspectorSynthView((Synth)synths.get(isynth), x, linewidth);
    synthviews.add(sv);
  }
  //EVENTREGIONS
  selectedenvregion=null;
  ArrayList envregions = new ArrayList();
  if(region.isEnvRegion)
    envregions.add((EnvRegion)region);
  if(envregions.size()==1)
    selectedenvregion = (Region)envregions.get(0);
  for(int ienvregion=0; ienvregion<envregions.size(); ienvregion++)
  {
    InspectorEnvRegionView ev=new InspectorEnvRegionView((EnvRegion)envregions.get(ienvregion), x, linewidth);
    envregionviews.add(ev);
  }
  updateViewYPositions();
}
void updateViewYPositions()
{
  int y = int(pos.y+gui.margin.y)+50;
  for(int i=0; i<headerguielements.size(); i++)
  {
    GUIElement view = (GUIElement)headerguielements.get(i);
    view.pos.y=y;
    if(i!=1)//write typename next to id --- dirty but ok for now
      y+=view.size.y;
  }
  y+=30;
  if(audiosendviews.size()>0)
  {
    audiosendlabel_id.pos.y = y;
    audiosendlabel_level.pos.y = y;
    y+=lineheight;
  }
  for(int iaudiosendview=0; iaudiosendview<audiosendviews.size(); iaudiosendview++)
  {
    InspectorAudioSendView view = (InspectorAudioSendView) audiosendviews.get(iaudiosendview);
    view.pos.y=y;
    view.updateViewYPositions();
    y+=view.size.y;
  }
  y+=30;
  for(int isynthview=0; isynthview<synthviews.size(); isynthview++)
  {
    InspectorSynthView view = (InspectorSynthView) synthviews.get(isynthview);
    view.pos.y=y;
    view.updateViewYPositions();
    y+=view.size.y+20;
  }
  for(int iopview=0; iopview<opviews.size(); iopview++)
  {
```

```
      InspectorOPView view = (InspectorOPView) opviews.get(iopview);
      view.pos.y=y;
      view.updateViewYPositions();
      y+=view.size.y+20;
    }
    for(int ienvregionview=0; ienvregionview<envregionviews.size(); ienvregionview++)
    {
      InspectorEnvRegionView view = (InspectorEnvRegionView) envregionviews.get(ienvregionview);
      view.pos.y=y;
      view.updateViewYPositions();
      y+=view.size.y+20;
    }
  }
  void run()
  {
    pushMatrix();
    translate(0,0,2);
    stroke(.4);
    fill(gui.backgroundcolor);
    rect(pos.x,pos.y,size.x,size.y);
    if(region==null)
    {
      popMatrix();
      return;
    }
    recordbutton.run();
    for(int iheaderguielement=0; iheaderguielement<headerguielements.size(); iheaderguielement++)
    {
      GUIElement hge=(GUIElement)headerguielements.get(iheaderguielement);
      hge.run();
    }
    if(audiosendviews.size()>0)
    {
      stroke(0,0,0,.8);
      fill(0,0,0,.4);
      InspectorAudioSendView firstaudiosendview = (InspectorAudioSendView) audiosendviews.get(0);
      rect(pos.x+gui.margin.x,firstaudiosendview.pos.y-20, size.x, audiosendviews.size()*firstaudiosendview.size.y+5+20);
      audiosendlabel_id.run();
      audiosendlabel_level.run();
    }
    for(int iaudiosendview=0; iaudiosendview<audiosendviews.size(); iaudiosendview++)
    {
      InspectorAudioSendView view = (InspectorAudioSendView) audiosendviews.get(iaudiosendview);
      view.run();
    }
    for(int isynthview=0; isynthview<synthviews.size(); isynthview++)
    {
      InspectorSynthView sv=(InspectorSynthView)synthviews.get(isynthview);
      sv.run();
    }
    for(int iopview=0; iopview<opviews.size(); iopview++)
    {
      InspectorOPView ov=(InspectorOPView)opviews.get(iopview);
      ov.run();
    }
    for(int ienvregionview=0; ienvregionview<envregionviews.size(); ienvregionview++)
    {
      InspectorEnvRegionView view = (InspectorEnvRegionView) envregionviews.get(ienvregionview);
      view.run();
    }
    popMatrix();
  }
};

public class InspectorSynthView extends GUIElement{
  Synth synth;
  ArrayList paramviews;
  GUIDropMenuButton changesynthbutton;
  InspectorSynthView(Synth _synth, float posx, float sizex)
  {
    synth=_synth;
    pos=new Coord(posx,0);
    size=new Coord(sizex, 30+synth.params.size()*gui.inspector.lineheight);
    paramviews = new ArrayList();
    changesynthbutton = new GUIDropMenuButton("Synth", synthdatabase.getSynthNames(), synthdatabase.getIndexOf(synth.name), color(0),color(.1,.5,.3),co
    int y = int(pos.y)+gui.inspector.lineheight;
    for(int iparam=0; iparam<synth.params.size(); iparam++)
      if(synth.getParam(iparam).type=='a')
        paramviews.add(new InspectorAudioParamView(synth.getParam(iparam), pos.x, gui.inspector.linewidth));
      else
        paramviews.add(new InspectorNumberParamView(synth.getParam(iparam), pos.x, gui.inspector.linewidth));
  }
  void run()
  {
    stroke(0,0,0,.8);
    fill(0,0,0,.4);
    rect(pos.x,pos.y, size.x,size.y-1);
    changesynthbutton.run();
    if(changesynthbutton.answer>=0)
    {
      Synth newsynth=synthdatabase.getSynth(changesynthbutton.answer);
      SynthRegion region=synth.region;
      region.setSynth(newsynth);
      changesynthbutton.answer=-1;
      gui.inspector.refresh();
      return;
    }
    for(int iparamview=0;iparamview<paramviews.size(); iparamview++)
    {
```

```
      InspectorParamView pv= (InspectorParamView) paramviews.get(iparamview);
      pv.run();
    }
  }
  void updateViewYPositions()
  {
    float y=pos.y+20;
    changesynthbutton.pos.y=pos.y;
    for(int iparamview=0;iparamview<paramviews.size(); iparamview++)
    {
      InspectorParamView view = (InspectorParamView)paramviews.get(iparamview);
      view.setPosY(y);
      view.updateViewYPositions();
      y+=view.size.y;
    }
    size.y=y-pos.y+8;
  }
  boolean mouseOver()
  {
    return mouseX>pos.x && mouseX<pos.x+size.x && mouseY>pos.y && mouseY<pos.y+size.y;
  }
};

abstract public class InspectorParamView extends GUIElement{
  GUIButton mainbutton;
  Param param;
  Coord pos, size;
  abstract void updateViewYPositions();
  void setPosY(float y)
  {
    pos.y=y;
    mainbutton.pos.y=y;
  }
};

public class InspectorNumberParamView extends InspectorParamView{
  InspectorNumberParamView(Param _param, float posx, float sizex)
  {
    param=_param;
    pos=new Coord(posx,0);
    size=new Coord(sizex,gui.inspector.lineheight);
    mainbutton=new GUIFloatEntryButton(((NumberParam)param).fieldtarget, param.name, color(0,0,1), color(0,0,.3), color(0,0,.5), pos, size);
  }
  void run()
  {
    //    display();
    mainbutton.run();
  }
  void updateViewYPositions()
  {
    final float y=pos.y+2;
    mainbutton.pos.y= y+gui.inspector.lineheight*0;
  }
};

public class InspectorAudioParamView extends InspectorParamView{
  boolean more=false;
  int inputmode=0;//0=float 1=audio 2=envregion
  GUIButton autobutton;
  InspectorAudioParamView(Param _param, float posx, float sizex)
  {
    param=_param;
    pos=new Coord(posx,0);
    size=new Coord(sizex,gui.inspector.lineheight);
    color fgColor=color(0,0,1);
    color bgColorNormal=color(0,0,1,.3);
    color bgColorMouseOver=color(0,0,1,.5);
    mainbutton=new GUIFloatOrRegionEntryButton(((AudioParam)param).fieldtarget, ((AudioParam)param).name, fgColor, bgColorNormal, bgColorMouseOver, new
    autobutton=new GUIButton("AUTO", fgColor, bgColorNormal, bgColorMouseOver, new Coord(pos.x+sizex-35,0), new Coord(35, gui.inspector.lineheight));
  }
  void run()
  {
    stroke(0,0,0,.8);
    fill(0,0,0,.4);
    mainbutton.run();
    if(mainbutton.wasclicked)
    {
      mainbutton.wasclicked=false;
    }
    autobutton.run();
    if(autobutton.wasclicked)
    {
      SynthRegion region=((AudioParam)param).synth.region;
      autobutton.wasclicked=false;
      EnvRegion er = region.newChildEnv("auto_"+param.name, 1);
      ((AudioParam)param).fieldtarget.setID(er.id);
      gui.inspector.updateViews(er);
    }
    if(((AudioParam)param).fieldtarget.isrouting)
    {
      Region r=param.synth.region.getRegionByID(((AudioParam)param).fieldtarget.getID());
      if(r!=null)
      {
        stroke(0,0,1,.7);
        line(r.view.pos.x+r.view.size.x, r.view.pos.y+7, pos.x,pos.y+7);
      }
    }
  }
  void updateViewYPositions()
  {
    final float y=pos.y+2;
    mainbutton.pos.y = autobutton.pos.y = y+gui.inspector.lineheight*0;
```

```
  }
};

public class InspectorOPView extends GUIElement{
  OP op;
  GUIDropMenuButton opname, applytype, channel;
  GUIFloatEntryButton arg;
  FloatInputFieldTarget argtarget;
  InspectorOPView(OP _op, float posx, float sizex)
  {
    op=_op;
    pos=new Coord(posx,0);
    size=new Coord(sizex, 20+4*gui.inspector.lineheight);
    Coord buttonsize=new Coord(sizex, gui.inspector.lineheight);
    //    int y = int(pos.y)+gui.inspector.lineheight;
    color fgColor=color(0,0,1);
    color bgColorNormal=color(.35,1,1,.2);
    color bgColorMouseOver=color(.35,1,1,.4);
    opname=new GUIDropMenuButton("Operator", opnames, op.opnum, fgColor, bgColorNormal, bgColorMouseOver, new Coord(pos.x,0), buttonsize, false);
    int applytypenumber=op.applytoregions ? (op.applytobreakpoints ? 2 : 1) :
    0;
    applytype=new GUIDropMenuButton("Apply to...", opapplytypenames, applytypenumber, fgColor, bgColorNormal, bgColorMouseOver, new Coord(pos.x,0), but
    channel=new GUIDropMenuButton("Channel", opchannelnames, op.channel, fgColor, bgColorNormal, bgColorMouseOver, new Coord(pos.x,0), buttonsize, fals
    argtarget=new FloatInputFieldTarget(op.arg);
    arg=new GUIFloatEntryButton(argtarget, "Argument", fgColor, bgColorNormal, bgColorMouseOver, new Coord(pos.x,0), buttonsize);
  }
  void run()
  {
    stroke(0,0,0,.8);
    fill(0,0,0,.4);
    rect(pos.x,pos.y, size.x,size.y-1);
    fill(1);
    text("OP", pos.x, pos.y+15);
    opname.run();
    if(opname.answer>=0)
    {
      op.opnum=opname.answer;
      opname.answer=-1;
    }
    applytype.run();
    if(applytype.answer>=0)
    {
      switch(applytype.answer)
      {
      case 0:
        op.applytoregions=false;
        op.applytobreakpoints=true;
        break;
      case 1:
        op.applytoregions=true;
        op.applytobreakpoints=false;
        break;
      case 2:
        op.applytoregions=true;
        op.applytobreakpoints=true;
        break;
      }
      applytype.answer=-1;
    }
    channel.run();
    if(channel.answer>=0)
    {
      op.channel=channel.answer;
      channel.answer=-1;
    }
    arg.run();
    if(op.arg!=argtarget.getFloat())
      op.arg=argtarget.getFloat();
  }
  void updateViewYPositions()
  {
    final float y=pos.y+10;
    applytype.pos.y= y+gui.inspector.lineheight*1;
    channel.pos.y=   y+gui.inspector.lineheight*2;
    opname.pos.y=    y+gui.inspector.lineheight*3;
    arg.pos.y=       y+gui.inspector.lineheight*4;
  }
};


public class InspectorEnvRegionView extends GUIElement{
  EnvRegion region;
  EnvRegionSelectButton selectbutton;
  GUIFloatEntryButton interpolation;
  InspectorEnvRegionView(EnvRegion _region, float posx, float sizex)
  {
    region=_region;
    pos=new Coord(posx,0);
    size=new Coord(sizex, 8+8*gui.inspector.lineheight);
    Coord buttonsize=new Coord(sizex, gui.inspector.lineheight);
    color fgColor=color(0,0,1);
    color bgColorNormal=color(.15,1,1,.2);
    color bgColorMouseOver=color(.15,1,1,.4);
    color bgColorSelected=color(.15,1,1,.6);
    selectbutton=new EnvRegionSelectButton("Breakpoints in "+region.name(), fgColor, bgColorNormal, bgColorMouseOver, bgColorSelected, new Coord(pos.x,
    interpolation=new GUIFloatEntryButton(region.interpolation, "Interpolation", fgColor, bgColorNormal, bgColorMouseOver, new Coord(pos.x,0), buttonsi
  }
  public class EnvRegionSelectButton extends GUIButton{
    color backColorSelected;
    EnvRegionSelectButton(String _title, color _foreColor, color _backColorNormal, color _backColorMouseover, color _backColorSelected, Coord _pos, Coo
    {
```

```
      super(_title, _foreColor, _backColorNormal, _backColorMouseover, _pos, _size);
      backColorSelected=_backColorSelected;
    }
    void click()
    {
      gui.inspector.selectedenvregion=region;
      backColorNormal=backColorSelected;
    }
    void display()
    {
      noStroke();
      if(gui.inspector.envregionviews.size()==1)
        fill(backColorSelected);
      else
        fill(mouseOver() ? backColorMouseover : gui.inspector.selectedenvregion==region ? backColorSelected : backColorNormal);
      rect(pos.x,pos.y, size.x,size.y);
      fill(foreColor);
      text(title, pos.x, pos.y+size.y-4);
    }
  };
  void run()
  {
    stroke(0,0,0,.8);
    fill(0,0,0,.4);
    selectbutton.run();
    interpolation.run();
    if(selectbutton.wasclicked)
    {
      selectbutton.wasclicked=false;
      println("ja");
    }
    EnvRegionView view = (EnvRegionView)region.view;
    text("heavy Range:  "+int(view.heavyminimum*1000)/1000.0+" ... "+view.heavymaximum, pos.x, pos.y+4*gui.inspector.lineheight);
    text("light Range:  "+int(view.minimum*1000)/1000.0+" ... "+view.maximum, pos.x, pos.y+3*gui.inspector.lineheight);
    if(gui.canvas.draggingbreakpoint!=null)
    {
      Breakpoint e=gui.canvas.draggingbreakpoint;
      text("time: "+e.lightStartTime(), pos.x, pos.y+6*gui.inspector.lineheight);
      text("light value: "+e.lightValueAt(1), pos.x, pos.y+7*gui.inspector.lineheight);
      text("heavy value: "+e.getHeavyValueAt(1), pos.x, pos.y+8*gui.inspector.lineheight);
    }
  }
  void updateViewYPositions()
  {
    final float y=pos.y+2;
    selectbutton.pos.y= y+gui.inspector.lineheight*0;
    interpolation.pos.y=   y+gui.inspector.lineheight*1;
  }
};

public class InspectorAudioSendView extends GUIElement{
  AudioSend send;
  GUIIDEntryButton idbutton;
  GUIFloatEntryButton levelbutton;
  GUIButton removebutton;
  GUILabel regionname;
  InspectorAudioSendView(AudioSend _send, float posx, float sizex)
  {
    send=_send;
    pos=new Coord(posx,0);
    size=new Coord(sizex, gui.inspector.lineheight);
    Coord buttonsize=new Coord(sizex, gui.inspector.lineheight);
    color fgColor=color(0,0,1);
    color bgColorNormal=color(.35,1,1,.2);
    color bgColorMouseOver=color(.35,1,1,.4);
    color bgColorSelected=color(.35,1,1,.6);
    idbutton=new GUIIDEntryButton(send.destinationID, "", fgColor, bgColorNormal, bgColorMouseOver, new Coord(pos.x+size.x-129,0), new Coord(44, gui.in
    levelbutton=new GUIFloatEntryButton(send.level, "", fgColor, bgColorNormal, bgColorMouseOver, new Coord(pos.x+size.x-83,0), new Coord(31, gui.inspe
    removebutton=new GUIButton("remove", fgColor, color(0,1,1,.3), color(0,1,1), new Coord(pos.x+size.x-50,0), new Coord(50, gui.inspector.lineheight))
    Region destinationRegion;
    if(send.destinationID.getInt()<0)
      destinationRegion = send.getSourceRegion().parent;
    else
      destinationRegion = session.getRegionByID(send.destinationID.getInt());
    if(destinationRegion==null)
    {
      warn("illegal routing from region "+send.getSourceRegion().id+" to non-existent region id "+send.destinationID.getInt());
      send.destinationID.setInt(0);
      destinationRegion = session.rootregion;
    }
    String labeltext=destinationRegion.name();
    while(textWidth(labeltext)>size.x-129)
      labeltext=labeltext.substring(labeltext.length()-1);
    regionname = new GUILabel(labeltext,color(1),new Coord(pos.x,0),new Coord(0,gui.inspector.lineheight));
  }
  void run()
  {
    regionname.run();
    idbutton.run();
    levelbutton.run();
    removebutton.run();
    if(removebutton.wasclicked)
    {
      removebutton.wasclicked=false;
      send.getSourceRegion().removeAudioSend(send);
      gui.inspector.updateViews(gui.inspector.region);
    }
  }
  void updateViewYPositions()
  {
    regionname.pos.y = pos.y;
```

```
    idbutton.pos.y = pos.y;
    levelbutton.pos.y = pos.y;
    removebutton.pos.y = pos.y;
  }
};
```

masterview.pde

```
public class MasterView extends GUIElement{
  SessionPathView sessionpathview;
  SnowButton snowbutton;
  MasterView(Coord _pos, Coord _size)
  {
    pos=_pos;
    size=_size;
    sessionpathview = new SessionPathView(new Coord(pos.x+gui.margin.x, pos.y+gui.margin.y), new Coord(400,40));
    snowbutton=new SnowButton("SNOW", color(0,0,0), color(0,0,.5), color(0,0,.8), color(0,0,1),  new Coord(pos.x+size.x-45-gui.margin.x, pos.y+gui.marg
  }
  void run()
  {
    pushMatrix();
    translate(0,0,2);
    snowbutton.runsnow();
    stroke(.4);
    fill(gui.backgroundcolor);
    rect(pos.x,pos.y,size.x,size.y);
    sessionpathview.run();
    snowbutton.run();
    popMatrix();
  }
};
public class SessionPathView extends GUIElement{
  GUIStringEntryButton pathentrybutton;
  PathTarget pathtarget;
  //  GUILabel projectnamelabel;
  //  GUILabel versionnumberlabel;
  int lineheight=15;
  SessionPathView(Coord _pos, Coord _size)
  {
    pos=_pos;
    size=_size;
    pathtarget = new PathTarget("---nix");
    pathentrybutton = new GUIStringEntryButton(pathtarget, "sessionPath", color(0,0,0), color(.6,.5,1,.5), color(.6,.5,1,1), new Coord(gui.margin.x, gu
    //  projectnamelabel = new GUILabel("name blabla", color(0,0,1), new Coord(margin.x,margin.y), new Coord(150,lineheight));
    //  versionnumberlabel = new GUILabel("version blabla", color(0,0,1), new Coord(margin.x+200-80,margin.y), new Coord(80,lineheight));
  }
  void run()
  {
    stroke(0,0,0,.8);
    fill(0,0,0,.4);
    rect(pos.x,pos.y, size.x,size.y-1);
    pathentrybutton.run();
    fill(1);
    String namestring = session.loadedpath.equals("") ? "--- new session ---" : "projectName: "+session.name;
    String versionstring = "version: "+session.versionnumber;
    text(namestring, pos.x, pos.y+lineheight+15);
    text(versionstring, pos.x+size.x-textWidth(versionstring), pos.y+lineheight+15);
  }
  public class PathTarget extends StringInputFieldTarget{
    PathTarget(String _stringValue)
    {
      super(_stringValue);
    }
    void setString(String s)
    {
      stringValue=s;
      sessionmanager.load(s);
    }
    void setStringAndNothingElse(String s)
    {
      stringValue=s;
    }
  };
};

public class SnowButton extends GUIButton{
  boolean active;
  color backColorActive;
  ArrayList flakes;
  int pMouseX, ppMouseX;
  SnowButton(String _title, color _foreColor, color _backColorNormal, color _backColorMouseover, color _backColorActive, Coord _pos, Coord _size, float
  {
    super(_title, _foreColor, _backColorNormal, _backColorMouseover, _pos, _size);
    backColorActive=_backColorActive;
    flakes=new ArrayList();
    for(int i=0;i<300; i++)
      flakes.add(new SnowFlake(new Coord(random(snowwidth), random(-height,0))));
```

```
  }
  void click()
  {
    active = !active;
    if(active)
      clickON();
    else
      clickOFF();
  }
  void clickON()
  {
  }
  void clickOFF()
  {
    for(int i=0;i<flakes.size(); i++)
    {
      SnowFlake f=(SnowFlake) flakes.get(i);
      f.pos.y=random(-height,0);
    }
  }
  void display()
  {
    noStroke();
    fill(mouseOver() ? backColorMouseover : active ? backColorActive : backColorNormal);
    rect(pos.x,pos.y, size.x,size.y);
    fill(foreColor);
    text(title, pos.x+1, pos.y+size.y-4);
    ppMouseX=pMouseX;
    pMouseX=mouseX;
  }
  void runsnow()
  {
    if(!active)
      return;
    for(int i=0;i<flakes.size(); i++)
    {
      SnowFlake f=(SnowFlake) flakes.get(i);
      f.run();
    }
  }
  class SnowFlake{
    Coord pos,vel;
    float rotangle, rotvel;
    SnowFlake(Coord _pos)
    {
      pos=_pos;
      //      vel=new Coord(0,0);
      vel=new Coord(random(-5,5), random(-3,6));
      rotangle=random(TWO_PI);
      rotvel=random(.1);
    }
    void run()
    {
      vel.x+=random(-1,1)*.1;
      vel.x*=.99;
      vel.y+=random(-1,1)*.1 + .01;
      vel.y*=.99;
      rotangle+=random(-1,1)*.1;
      rotvel+=random(-1,1)*.1;
      rotvel*=.99;
      if(gui.canvas.draggingregionview!=null)
      {
        RegionView rv = gui.canvas.draggingregionview;
        if(pos.y>rv.pos.y && pos.y<rv.pos.y+rv.size.y)
          if(pos.x>rv.pos.x && pos.x<rv.pos.x+rv.size.x)
          {
            float mousevel = mouseX-gui.masterview.snowbutton.ppMouseX;
            vel.x+= constrain(mousevel, -5, 5) *.1;
            if(mousevel!=0)
              vel.x*=.9;
            vel.y+=random(-1,1)*.4 - .01;
            rotvel+=mousevel*.01;
            rotangle*=1+mousevel*.1;
          }
      }
if(transport.playing)
      {
        float cursordistance= abs(gui.canvas.seconds2pos(transport.curtime)-3 - pos.x);
        rotangle*=1+.5/max(4,cursordistance);
if(abs(rotangle)>3)
        rotangle*=.9;
        rotvel*=1+1/max(10,cursordistance);
        if(abs(rotvel)>2)
        rotvel*=.9;
        vel.x= gui.canvas.pixelspersecond/frameRate *2/max(1,cursordistance);
//        vel.x+=.2/max(10,cursordistance);
      }
      if(rotangle>PI*4)
      rotangle-=PI;
      if(rotangle<-PI*4)
      rotangle+=PI;
      pos.x+= vel.x +rotvel*cos(rotangle);
      pos.y+= vel.y -rotvel*sin(rotangle);
      if(pos.y>height)
      {
        pos.y=random(-height);
        vel.x*=.5;
        rotvel*=.5;
      }
      if(pos.x<gui.canvas.pos.x)
      {  pos.x=gui.inspector.pos.x+4;
        vel.x*=.5;
        rotvel*=.5;
```

```
      }
      if(pos.x>gui.inspector.pos.x+4)
      {  pos.x=gui.canvas.pos.x;
        vel.x*=.5;
        rotvel*=.5;
      }
      noStroke();
      fill(0,0,1, random(.4,.5));
      rect(pos.x,pos.y-1, 3,5);
      fill(0,0,1, random(.4,.5));
      rect(pos.x-2,pos.y, 7,3);
      rect(gui.canvas.pos.x+gui.canvas.size.x,0, gui.canvas.pos.x+gui.canvas.size.x+2,500);
    }
  };
};
```

op.pde

```
String []opnames={
  "pow","*","+","log","min","max","abs","quantize"};
String []opapplytypenames={
  "Breakpoints", "Regions", "Breakpoints and Regions"};
String []opchannelnames={
  "Time", "Y/Duration"};

int getOPNumByName(String name)
{
  for(int i=0;i<opnames.length;i++)
    if(name.equals(opnames[i]))
      return i;
  return -1;
}
public class OP{
  int opnum, channel;//channel 0 is starttime, channel 1 is duration/y
  boolean applytoregions, applytobreakpoints;
  float arg;
  Region region;
  OP(int _opnum, float _arg, int _channel, boolean _applytoregions, boolean _applytobreakpoints)
  {
    opnum=_opnum;
    arg=_arg;
    channel=_channel;
    applytoregions=_applytoregions;
    applytobreakpoints=_applytobreakpoints;
    if(opnum<0||opnum>=opnames.length)
      fail("illegal opnum "+opnum);
    if(applytoregions==false && applytobreakpoints==false)
      fail("either applytoregions or applytobreakpoints must be true");
  }
  String name()
  {
    return opnames[opnum];
  }
  float apply(float value)
  {
    //  "pow","mul","add","log","min","max","abs","qua"};
    switch(opnum)
    {
    case 0:
      return pow(value, arg);
    case 1:
      return value*arg;
    case 2:
      return value+arg;
    case 3:
      return log(value)/log(arg);
    case 4:
      return min(value,arg);
    case 5:
      return max(value,arg);
    case 6:
      return abs(value);
    case 7:
      return int(value/arg+.5) * arg;
    }
    fail("programming error: op num "+opnum+" not implemented");
    return 0;//never reached
  }
  XMLElement getXMLElement()
  {
    XMLElement result=new XMLElement("op");
    result.addAttribute("name", opnames[opnum]);
    result.addAttribute("channel", channel);
    result.addAttribute("arg", arg);
    result.addAttribute("applytoregions", applytoregions ? "true" : "false");
    result.addAttribute("applytobreakpoints", applytobreakpoints ? "true" : "false");
    return result;
  }
};
```

param.pde

```
abstract public class Param{
  Synth synth;
  //  GUITextInputFieldTarget fieldtarget;
  String name;
  char type;
  boolean typeIsString()
  {
    return type=='s' || type=='p';
  }
  boolean typeIsNumeric()
  {
    return type=='n' || type=='a';
  }
  abstract String getString();
  abstract void setString(String s);
  abstract XMLElement getXMLElement();
  abstract Param copy();
  void print()
  {
    println("param "+name+"  "+type+"  "+getString());
  }
  ////////////////////////////////////// SCRIPTING
  int getType()
  {
    return 0+type;
  }
};

public class StringParam extends Param {
  StringInputFieldTarget fieldtarget;
  StringParam(String _name, String _stringValue)
  {
    type='s';
    name=_name;
    fieldtarget=new StringInputFieldTarget(_stringValue);
  }
  String getString()
  {
    return fieldtarget.getString();
  }
  void setString(String s)
  {
    fieldtarget.setString(s);
  }
  XMLElement getXMLElement()
  {
    XMLElement result=new XMLElement("stringparam");
    result.addAttribute("name",name);
    result.addAttribute("value", getString());
    return result;
  }
  Param copy()
  {
    return new StringParam(name, getString());
  }
};

public class AudioParam extends Param {
  FloatOrRegionInputFieldTarget fieldtarget;
  AudioParam(String _name, FloatOrRegionInputFieldTarget _fieldtarget)
  {
    type='a';
    name=_name;
    fieldtarget=_fieldtarget;
  }
  AudioParam(String _name, float value, boolean isrouting)
  {
    type='a';
    name=_name;
    fieldtarget=new FloatOrRegionInputFieldTarget(value, isrouting);
  }
  float getFloat()
  {
    return fieldtarget.getFloat();
  }
  String getString()
  {
    return fieldtarget.getString();
  }
  void setString(String s)
  {
    fieldtarget.setString(s);
  }
  void setFloat(float x)
  {
    fieldtarget.setFloat(x);
  }
  XMLElement getXMLElement()
  {
    XMLElement result=new XMLElement("audioparam");
    result.addAttribute("name",name);
    result.addAttribute("value", fieldtarget.floatValue);
    result.addAttribute("isrouting", fieldtarget.isrouting ? 1 : 0);
    return result;
  }
  Param copy()
  {
```

```
    return new AudioParam(name, fieldtarget.floatValue, fieldtarget.isrouting);
  }
  ////////////////////////////////////// SCRIPTING
  int getIsRouting()
  {
    return fieldtarget.isrouting ? 1:0;
  }
  float getValue()
  {
    return fieldtarget.floatValue;
  }
};

public class NumberParam extends Param {
  FloatInputFieldTarget fieldtarget;
  NumberParam(String _name, float _floatValue)
  {
    type='n';
    name=_name;
    fieldtarget = new FloatInputFieldTarget(_floatValue);
  }
  void setFloat(float x)
  {
    fieldtarget.setFloat(x);
  }
  float getFloat()
  {
    return fieldtarget.getFloat();
  }
  String getString()
  {
    return fieldtarget.getString();
  }
    void setString(String s)
   {
   fieldtarget.setString(s);
   }
  XMLElement getXMLElement()
  {
    XMLElement result=new XMLElement("numberparam");
    result.addAttribute("name",name);
    result.addAttribute("value",getString());
    return result;
  }
  Param copy()
  {
    return new NumberParam(name, getFloat());
  }
};
```

region.pde

```
abstract public class ThingInTime{
  protected Region parent;
  protected FloatInputFieldTarget[]heavyvalues;
  abstract float lightStartTime();
  abstract float lightValueAt(int index);
  abstract XMLElement getXMLElement();
  int countValues()
  {
    return heavyvalues.length;
  }
  final float heavyStartTime()
  {
    return heavyvalues[0].getFloat();
  }
  final public void setHeavyStartTime(float t)
  {
    heavyvalues[0].setFloat(t);
  }
  float getHeavyValueAt(int index)
  {
    return heavyvalues[index].getFloat();
  }
  void setHeavyValueAt(int index, float value)
  {
    heavyvalues[index].setFloat(value);
  }
  FloatInputFieldTarget heavyValueAt(int index)
  {
    return heavyvalues[index];
  }
};

public class Breakpoint extends ThingInTime{
  Breakpoint(Region _parent, FloatInputFieldTarget[]_heavyvalues)
  {
    parent=_parent;
    heavyvalues=_heavyvalues;
```

```
  }
  float lightStartTime()
  {
    return parent.passFloatUpThroughOPs(heavyStartTime(), heavyStartTime(), true, 0);
  }
  float lightValueAt(int channel)
  {
    return parent.passFloatUpThroughOPs(getHeavyValueAt(channel), heavyStartTime(), true, channel);
  }
  XMLElement getXMLElement()
  {
    XMLElement result = new XMLElement("breakpoint");
    result.addAttribute("starttime",heavyStartTime());
    result.addAttribute("y1",getHeavyValueAt(1));
    return result;
  }
};

abstract public class Region extends ThingInTime{
  RegionWithChildren parent;
  String typename;
  int id;
  protected StringInputFieldTarget namefieldtarget;
  RegionView view;
  RegionWithChildren getParent;
  boolean isGroupRegion, isSynthRegion, isOPRegion, isEnvRegion;
  abstract void init();
  abstract void dragDownTree(float dragtime);
  abstract void destroy();
  abstract Region getDescendantByName(String the_name);
  abstract Region getDescendantByID(int the_id);
  float heavyDuration()
  {
    return heavyvalues[1].getFloat();
  }
  void setHeavyDuration(float t)
  {
    heavyvalues[1].setFloat(t);
  }
  float heavyEndTime()
  {
    return heavyvalues[0].getFloat()+heavyvalues[1].getFloat();
  }
  float lightStartTime()
  {
    if(parent!=null)
      return parent.passFloatUpThroughOPs(heavyStartTime(), heavyStartTime(), false, 0);
    return heavyStartTime();
  }
  float lightDuration()
  {
    if(parent!=null)
      return parent.passFloatUpThroughOPs(heavyDuration(), heavyStartTime(), false, 1);
    return heavyDuration();
  }
  float lightValueAt(int channel)
  {
    if(channel>1)
      fail("region.lightValueAt called with channel>1 . regions can only have 2 channels: 0=starttime 1=duration");
    return parent.passFloatUpThroughOPs(getHeavyValueAt(channel), heavyStartTime(), false, channel);
  }
  float lightEndTime()
  {
    return lightStartTime()+lightDuration();
  }
  String name()
  {
    return namefieldtarget.getString();
  }
  void setName(String s)
  {
    namefieldtarget.setString(s);
  }
  Region getRegionByID(int the_id)
  {
    if(id==the_id)
      return this;
    if(the_id==-1)
      return parent;
    if(the_id==-2)
      return session.rootregion;
    return session.getRegionByID(the_id);
  }
  float passFloatUpThroughOPs(float inVal, float time, boolean comesFromABreakpoint, int channel)
  {
    if(parent!=null)
      return parent.passFloatUpThroughOPs(inVal, time, comesFromABreakpoint, channel);
    return inVal;
  }
  void writeHeadToXML(XMLElement element)
  {
    element.addAttribute("name",name());
    element.addAttribute("id",id);
    element.addAttribute("starttime",heavyStartTime());
    element.addAttribute("duration",heavyDuration());
  }
  private void newParentCommon(RegionWithChildren newparent)
  {
    newparent.init();
    newparent.addChild(this);
    if(newparent.parent==null)
```

```
    {
      session.rootregion=newparent;
      gui.canvas.focusSession();
    }
    else
    {
      RegionWithChildren oldparent = newparent.parent;
      oldparent.replaceChild(this,newparent);
      newparent.addAudioSend(new AudioSend(-1, 1));
    }
    session.updateGenerations();
    newparent.view.updateXYDownTree(newparent.view.pos.y, newparent.view.size.y);
    gui.inspector.updateViews(newparent);
    displayneedsupdate=true;
    session.markDirty();
  }
  void newParentGroup(String name)
  {
    newParentGroup(name,heavyStartTime(),heavyDuration());
  }
  void newParentGroup(String name, float starttime, float duration)
  {
    GroupRegion newparent=new GroupRegion(parent,name, session.nextFreeID(), makeFloatInputFieldTargetArray(starttime,duration));
    newParentCommon(newparent);
  }
  void newParentSynth(String name)
  {
    newParentSynth(name,heavyStartTime(),heavyDuration());
  }
  void newParentSynth(String name, float starttime, float duration)
  {
    SynthRegion newparent=new SynthRegion(parent,name, session.nextFreeID(), makeFloatInputFieldTargetArray(starttime,duration));
    Synth synth=synthdatabase.create("BPF");
    if(synth==null)
      return;
    newparent.setSynth(synth);
    newParentCommon(newparent);
  }
  void newParentOP(String name)
  {
    newParentOP(name,heavyStartTime(),heavyDuration());
  }
  void newParentOP(String name, float starttime, float duration)
  {
    OPRegion newparent=new OPRegion(parent,name, session.nextFreeID(), makeFloatInputFieldTargetArray(starttime,duration));
    newparent.setOP(new OP(2,0,1,false,true));
    newParentCommon(newparent);
  }
  void moveUp()
  {
    if(parent==null || parent.parent==null)
      return;
    RegionWithChildren grandpa = parent.parent;
    parent.removeChild(this);
    grandpa.addChild(this);
  }
  ////////////////////////// SCRIPTING ONLY
  void bla()
  {
    println("blablabla i am a region. my name is "+name());
  }
  int getID()
  {
    return id;
  }
  int isGroup() // OSC has trouble with boolean values, so use int
  {
    return isGroupRegion ? 1 : 0;
  }
  int isSynth()
  {
    return isSynthRegion ? 1 : 0;
  }
  int isOP()
  {
    return isOPRegion ? 1 : 0;
  }
  int isEnv()
  {
    return isEnvRegion ? 1 : 0;
  }
  int getParentID()
  {
    if(parent==null)
      return -1;
    return parent.id;
  }
};


abstract public class RegionWithChildren extends Region{
  protected ArrayList children;
  protected ArrayList audiosends;
  Fade fade;
  void init()
  {
    println("RegionWithChildren "+name()+" init");
    children = new ArrayList();
    view = new RegionWithChildrenView(this);
    audiosends = new ArrayList();
    fade = new Fade(.001,1,.001,1);
```

```
}
int countChildren()
{
  return children.size();
}
Region getChild(int index)
{
  return (Region) children.get(index);
}
Region getDescendantByID(int the_id)
{
  if(id==the_id)
    return this;
  for(int ichild=0;ichild<children.size();ichild++)
  {
    Region found = getChild(ichild).getDescendantByID(the_id);
    if(found!=null)
      return found;
  }
  return null;
}
Region getDescendantByName(String the_name)
{
  if(name().equals(the_name))
    return this;
  for(int ichild=0;ichild<children.size();ichild++)
  {
    Region found = getChild(ichild).getDescendantByName(the_name);
    if(found!=null)
      return found;
  }
  return null;
}
void addChild(Region child)
{
  children.add(child);
  child.parent=this;
}
void removeChild(int index)
{
  children.remove(index);
}
void removeChild(Region region)
{
  children.remove(region);
}
void removeAllChildren()
{
  children=new ArrayList();
}
void replaceChild(Region existingregion, Region newregion)
{
  int index = children.indexOf(existingregion);
  children.set(index, newregion);
}
int countAudioSends()
{
  return audiosends.size();
}
AudioSend getAudioSend(int index)
{
  return (AudioSend) audiosends.get(index);
}
void addAudioSend(AudioSend send)
{
  send.setSourceRegion(this);
  audiosends.add(send);
  if(gui.inspector.region==this)
    gui.inspector.updateViews(this);
}
void removeAudioSend(AudioSend send)
{
  send.setSourceRegion(null);
  audiosends.remove(send);
}
void setFade(Fade f)
{
  fade=f;
}
void collectSynthsUpTree(ArrayList result)
{
  if(parent!=null)
    ((RegionWithChildren)parent).collectSynthsUpTree(result);
}
int countAncestors(int sum)
{
  for(int ichild=0;ichild<countChildren();ichild++)
    if(getChild(ichild).isEnvRegion)
      sum++;
    else
      sum += ((RegionWithChildren)getChild(ichild)).countAncestors(sum);
  return sum;
}
void collectSenders(RegionWithChildren region, ArrayList result)//search the tree for regions which send audio to this one
{
  for(int isend=0; isend<region.countAudioSends(); isend++)
  {
    AudioSend send = region.getAudioSend(isend);
    if(send.destinationID.getInt()==id || (send.destinationID.getInt()==-1 && region.parent==this))
      result.add(region);
  }
  for(int ichild=0;ichild<region.countChildren();ichild++)
    if(!region.getChild(ichild).isEnvRegion)
```

```
      collectSenders((RegionWithChildren)region.getChild(ichild), result);
}
boolean sortChildrenByTime()
{
  boolean changedanything=false;
  if(children.size()<2)
    return false;
  ThingInTime thing;
  ThingInTime prev = (ThingInTime) children.get(0);
  for(int ithing=1;ithing<children.size();ithing++)
  {
    thing=(ThingInTime) children.get(ithing);
    if(thing.heavyStartTime()<prev.heavyStartTime())
    {
      children.set(ithing, prev);
      children.set(ithing-1, thing);
      changedanything=true;
    }
    prev=thing;
  }
  if(changedanything)
    return sortChildrenByTime();
  return false;
}
void writeAudioSendsFadesAndChildrenToXML(XMLElement element)
{
  for(int isend=0;isend<audiosends.size();isend++)
    element.addChild(getAudioSend(isend).getXMLElement());
  for(int ichild=0;ichild<children.size();ichild++)
    element.addChild(getChild(ichild).getXMLElement());
  element.addChild(fade.getXMLElement());
}
void destroy()
{
  debug("TODO destroy");
}
private void newChildCommon(Region child)
{
  child.init();
  if(!child.isEnvRegion)
    ((RegionWithChildren)child).addAudioSend(new AudioSend(-1, 1));
  addChild(child);
  session.updateGenerations();
  view.updateXYDownTree(view.pos.y, view.size.y);
  gui.canvas.updateXY();
  displayneedsupdate=true;
  session.markDirty();
}
GroupRegion newChildGroup(String name)
{
  return newChildGroup(name,heavyStartTime(),heavyDuration());
}
GroupRegion newChildGroup(String name, float starttime, float duration)
{
  GroupRegion child=new GroupRegion(this,name, session.nextFreeID(), makeFloatInputFieldTargetArray(starttime,duration));
  newChildCommon(child);
  return child;
}
OPRegion newChildOP(String name)
{
  return newChildOP(name,heavyStartTime(),heavyDuration());
}
OPRegion newChildOP(String name, float starttime, float duration)
{
  OPRegion child=new OPRegion(this,name, session.nextFreeID(), makeFloatInputFieldTargetArray(starttime,duration));
  child.setOP(new OP(2,0,1,false,true));
  newChildCommon(child);
  return child;
}
SynthRegion newChildSynth(String name, String synthname)
{
  return newChildSynth(name,heavyStartTime(),heavyDuration(), synthname);
}
SynthRegion newChildSynth(String name, float starttime, float duration, String synthname)
{
  SynthRegion child=new SynthRegion(this,name, session.nextFreeID(), makeFloatInputFieldTargetArray(starttime,duration));
  Synth newsynth=synthdatabase.create(synthname);
  if(newsynth==null)
    return null;
  child.setSynth(newsynth);
  newChildCommon(child);
  return child;
}
EnvRegion newChildEnv(String name, float interpolation)
{
  return newChildEnv(name, heavyStartTime(), heavyDuration(), interpolation);
}
EnvRegion newChildEnv(String name, float starttime, float duration, float interpolation)
{
  EnvRegion child=new EnvRegion(this, name, session.nextFreeID(), makeFloatInputFieldTargetArray(starttime,duration), interpolation);
  newChildCommon(child);
  return child;
}
void dragDownTree(float dragtime)
{
  switch(gui.canvas.dragmode)
  {
  case 0://drag region with content
    setHeavyStartTime(heavyStartTime()+dragtime);
    for(int ichild=0;ichild<countChildren();ichild++)
      getChild(ichild).dragDownTree(dragtime);
```

```
      break;
    case 1://drag left edge
      float endtime=heavyEndTime();
      setHeavyStartTime(min(heavyStartTime()+dragtime, endtime));
      setHeavyDuration(endtime-heavyStartTime());
      break;
    case 2://drag right edge
      setHeavyDuration(max(0, heavyDuration()+dragtime));
      break;
    }
  }
  /////////////////////////// SCRIPTING ONLY
  float fadeInTime()
  {
    return fade.intime.getFloat();
  }
  void setFadeInTime(float x)
  {
    fade.intime.setFloat(max(0,x));
  }
  float fadeInCurve()
  {
    return fade.incurve.getFloat();
  }
  void setFadeInCurve(float x)
  {
    fade.incurve.setFloat(x);
  }
  float fadeOutTime()
  {
    return fade.outtime.getFloat();
  }
  void setFadeOutTime(float x)
  {
    fade.outtime.setFloat(max(0,x));
  }
  float fadeOutCurve()
  {
    return fade.outcurve.getFloat();
  }
  void setFadeOutCurve(float x)
  {
    fade.outcurve.setFloat(x);
  }
};

public class GroupRegion extends RegionWithChildren{
  GroupRegion(RegionWithChildren _parent, String _name, int _id, FloatInputFieldTarget[]_heavyvalues)
  {
    parent=_parent;
    namefieldtarget=new StringInputFieldTarget(_name);
    id=_id;
    heavyvalues=_heavyvalues;
    isGroupRegion=true;
    typename="group";
  }
  XMLElement getXMLElement()
  {
    XMLElement result = new XMLElement("groupregion");
    writeHeadToXML(result);
    writeAudioSendsFadesAndChildrenToXML(result);
    return result;
  }
};

public class SynthRegion extends RegionWithChildren{
  private Synth synth;
  SynthRegion(RegionWithChildren _parent, String _name, int _id,FloatInputFieldTarget[]_heavyvalues)
  {
    parent=_parent;
    namefieldtarget=new StringInputFieldTarget(_name);
    id=_id;
    heavyvalues=_heavyvalues;
    isSynthRegion=true;
    typename="synth";
  }
  void setSynth(Synth s)
  {
    if(synth!=null && synthdatabase.getIndexOf(synth.name)>=0)//if region was named by a synth, rename it
    {
      namefieldtarget.setString(s.name);
      println("renamed");
    }
    synth=s;
    synth.region=this;
  }
  Synth getSynth()
  {
    return synth;
  }
  void collectSynthsUpTree(ArrayList result)
  {
    result.add(synth);
    if(parent!=null)
      parent.collectSynthsUpTree(result);
  }
  XMLElement getXMLElement()
  {
    XMLElement result = new XMLElement("synthregion");
    writeHeadToXML(result);
    result.addChild(synth.getXMLElement());
```

```
    writeAudioSendsFadesAndChildrenToXML(result);
    return result;
  }
  ////////////////////////// SCRIPTING ONLY
  void bla()
  {
    println("blablabla i am a SynthRegion. my name is "+name());
  }
  String getSynthName()
  {
    return synth.name;
  }
  void setParam(int index, float value)
  {
    synth.setParam(index, value);
  }
};

public class OPRegion extends RegionWithChildren{
  private OP op;
  OPRegion(RegionWithChildren _parent, String _name, int _id, FloatInputFieldTarget[]_heavyvalues)
  {
    parent=_parent;
    namefieldtarget=new StringInputFieldTarget("OPRegion unnamed");
    id=_id;
    heavyvalues=_heavyvalues;
    isOPRegion=true;
    typename="op";
  }
  void setOP(OP _op)
  {
    if(op!=null)
      warn("setOP overwrote OP "+op.name()+" with OP "+_op.name());
    namefieldtarget.setString(_op.name());
    op=_op;
    op.region=this;
  }
  OP getOP()
  {
    return(op);
  }
  float passFloatUpThroughOPs(float inVal, float time, boolean comesFromABreakpoint, int channel)
  {
    if(time>=lightStartTime() && time<=lightEndTime() && channel==op.channel && ((comesFromABreakpoint && op.applytobreakpoints) || (!comesFromABreakpo
      inVal=op.apply(inVal);
    if(parent!=null)
      return parent.passFloatUpThroughOPs(inVal, time, comesFromABreakpoint, channel);
    return inVal;
  }
  XMLElement getXMLElement()
  {
    XMLElement result = new XMLElement("opregion");
    writeHeadToXML(result);
    result.addChild(op.getXMLElement());
    writeAudioSendsFadesAndChildrenToXML(result);
    return result;
  }
  ThingInTime copy()
  {
    return null;
  }
};

public class EnvRegion extends Region{
  private ArrayList breakpoints;
  private FloatInputFieldTarget interpolation;
  EnvRegion(RegionWithChildren _parent, String _name, int _id, FloatInputFieldTarget[]_heavyvalues, float _interpolation)
  {
    parent=_parent;
    namefieldtarget=new StringInputFieldTarget(_name);
    id=_id;
    heavyvalues=_heavyvalues;
    interpolation= new FloatInputFieldTarget(_interpolation);
    isEnvRegion=true;
    typename="breakpoints";
  }
  void init(){
    breakpoints = new ArrayList();
    view = new EnvRegionView(this);
  }
  int countBreakpoints()
  {
    return breakpoints.size();
  }
  Region getDescendantByName(String the_name)
  {
    if(name().equals(the_name))
      return this;
    return null;
  }
  Region getDescendantByID(int the_id)
  {
    if(id==the_id)
      return this;
    return null;
  }
  void addBreakpoint(Breakpoint child)
  {
    breakpoints.add(child);
  }
  Breakpoint getBreakpoint(int index)
  {
```

```
    return (Breakpoint) breakpoints.get(index);
  }
  void removeBreakpoint(int index)
  {
    if(index<0||index>=breakpoints.size())
      fail("removeBreakpoint got wrong index "+index+".   breakpoints.size="+breakpoints.size());
    breakpoints.remove(index);
  }
  void removeAllBreakpoints()
  {
    breakpoints=new ArrayList();
  }
  void removeBreakpointsInTimeRange(float firsttime, float lasttime)
  {
    for(int ibreakpoint=breakpoints.size()-1; ibreakpoint>=0; ibreakpoint--)
      if(getBreakpoint(ibreakpoint).heavyStartTime()>=firsttime && getBreakpoint(ibreakpoint).heavyStartTime()<=lasttime)
        removeBreakpoint(ibreakpoint);
  }
  boolean sortBreakpointsByTime()
  {
    boolean changedanything=false;
    if(breakpoints.size()<2)
      return false;
    ThingInTime thing;
    ThingInTime prev = (ThingInTime) breakpoints.get(0);
    for(int ithing=1;ithing<breakpoints.size();ithing++)
    {
      thing=(ThingInTime) breakpoints.get(ithing);
      if(thing.heavyStartTime()<prev.heavyStartTime())
      {
        breakpoints.set(ithing, prev);
        breakpoints.set(ithing-1, thing);
        changedanything=true;
      }
      prev=thing;
    }
    if(!changedanything)
      return false;
    while(sortBreakpointsByTime())
      ;
    return true;
  }
  void dragDownTree(float dragtime)
  {
    switch(gui.canvas.dragmode)
    {
    case 0://drag region with content
      setHeavyStartTime(heavyStartTime()+dragtime);
      for(int ibreakpoint=0;ibreakpoint<countBreakpoints();ibreakpoint++)
        getBreakpoint(ibreakpoint).setHeavyStartTime(getBreakpoint(ibreakpoint).heavyStartTime()+dragtime);
      break;
    case 1://drag left edge
      float endtime=heavyEndTime();
      setHeavyStartTime(min(heavyStartTime()+dragtime, endtime));
      setHeavyDuration(endtime-heavyStartTime());
      break;
    case 2://drag right edge
      setHeavyDuration(max(0, heavyDuration()+dragtime));
      break;
    }
  }
  XMLElement getXMLElement()
  {
    XMLElement result = new XMLElement("envregion");
    writeHeadToXML(result);
    result.addAttribute("interpolation",interpolation.getFloat());
    for(int ibreakpoint=0;ibreakpoint<breakpoints.size();ibreakpoint++)
      result.addChild(getBreakpoint(ibreakpoint).getXMLElement());
    return result;
  }
  void destroy()
  {
    debug("TODO destroy");
  }
  Breakpoint newBreakpoint(float time, float y01)
  {
    Breakpoint b=new Breakpoint(this, makeFloatInputFieldTargetArray(time, y01));
    addBreakpoint(b);
    displayneedsupdate = sortBreakpointsByTime();
    session.markDirty();
    return b;
  }
  public void bla()
  {
    System.out.println("blablabla i am an EnvRegion. my name is "+name());
  }
  ////////////////////////// SCRIPTING ONLY
  float getInterpolation()
  {
    return interpolation.getFloat();
  }
};
```

regionview.pde

```
abstract public class RegionView{
  Region region;
  Coord size,pos;
  float headsizex;
  int headsizey=15, minimumheadsizex=30;
  int generation;
  boolean selected;
  abstract void updateXYDownTree(float posy, float sizey);
  abstract void run();
  abstract float zoomY();
  //  abstract void actionmenu();
  void actionmenu()
  {
    if(((keycodesdown[CONTROL] && clickpending==LEFT) || clickpending==RIGHT) && mouseOverHead())
    {
      clickpending=0;
      gui.canvas.regionActionMenu.activate(region, new Coord(mouseX-70,pos.y));
    }
    if(keycodesdown[SHIFT] && clickpending==LEFT && mouseOverHead())
    {
      clickpending=0;
      selected=!selected;
    }
  }
  void updateGeneration(int _generation)
  {
    generation=_generation;
  }
  void updateX()
  {
    pos.x = gui.canvas.seconds2pos(region.heavyStartTime());
    size.x = region.heavyDuration() * gui.canvas.pixelspersecond;
    headsizex = max(minimumheadsizex, size.x);
  }
  int countSelectedSiblings()
  {
    ArrayList affe=new ArrayList();
    collectSelectedSiblings(affe);
    return affe.size();
  }
  void collectSelectedSiblings(ArrayList result)
  {
    if(region.parent==null)
      return;
    ((RegionWithChildrenView)region.parent.view).collectSelectedChildren(result);
  }
  void unselectAncestors()
  {
    selected=false;
  }
  final void move()
  {
    updateX();
    boolean dragginganotherregion = gui.canvas.draggingregionview!=null && gui.canvas.draggingregionview!=this;
    //start dragging region?
    if(!keycodesdown[CONTROL] && clickpending==LEFT && mouseOverHead() && gui.canvas.draggingregionview!=this)
    {
      if(mouseOverHeadCenter())
      {
        gui.inspector.updateViews(region);
      }
      gui.canvas.draggingregionview=this;
      //      gui.canvas.dragStartPosX=mouseX;
      gui.canvas.dragPointOnHeadFromLeft=mouseX-pos.x;
      gui.canvas.dragPointOnHeadFromRight=pos.x+headsizex-mouseX;
      gui.canvas.dragmode = (mouseOverHeadLeft() ? 1 : (mouseOverHeadRight() ? 2 : 0));
      displayneedsupdate=true;
      clickpending=0;
    }
    //dragging this region
    if(gui.canvas.draggingregionview==this)
    {
      float dragpixels=0;
      if(mousePressed)
      {
        displayneedsupdate=true;
        if(gui.canvas.dragmode==2)
          dragpixels=mouseX-pos.x-headsizex+gui.canvas.dragPointOnHeadFromRight;
        else
          dragpixels=mouseX-gui.canvas.dragPointOnHeadFromLeft-pos.x;
      }
      else//drop
      {
        gui.canvas.draggingregionview=null;
        // gui.inspector.updateViews(region);
        if(dragpixels!=0)
          session.markDirty();
        if(region.parent!=null)
        {
          region.parent.view.updateXYDownTree(region.parent.view.pos.y, region.parent.view.size.y);
          //region.parent.sortChildrenByTime();
        }
      }
      region.dragDownTree(dragpixels/gui.canvas.pixelspersecond);
    }
  }
  void displayBody()
```

```
{
  updateX();
  boolean dragging = gui.canvas.draggingregionview==this;
  boolean inspected = gui.inspector.region==region;
  boolean dragginganotherregion = gui.canvas.dragmode>=0 &! dragging;
  float colorhue=1-(.3+generation*.3)%1;
  float opacityleft =   dragginganotherregion ? .3 : mouseOverHeadLeft() ?.5:.3;
  float opacitycenter = dragginganotherregion ? .3 : mouseOverHeadCenter() ?.5:.3;
  float opacityright =  dragginganotherregion ? .3 : mouseOverHeadRight() ?.5:.3;
  if(inspected)
    opacitycenter=.7;
  if(dragging)
  {
    switch(gui.canvas.dragmode)
    {
    case 0:
      opacitycenter=.8;
      gui.canvas.lineVertical(pos.x, color(1));
      break;
    case 1:
      opacitycenter=(inspected ? .8 : .3);
      opacityleft=1;
      gui.canvas.lineVertical(pos.x, color(1));
      break;
    case 2:
      opacitycenter=(inspected ? .8 : .3);
      opacityright=1;
      gui.canvas.lineVertical(pos.x+headsizex, color(1));
      break;
    }
  }
  noStroke(); //paint color body
  if((mouseOverHeadLeft() && gui.canvas.dragmode<0) || (gui.canvas.dragmode==1 && dragging))
  {
    if(size.x==headsizex)
    {
      fill(colorhue,1,opacityleft);
      rect(pos.x,pos.y,headedgewidth(),size.y-1);
      fill(colorhue,1,opacitycenter);
      rect(pos.x+headedgewidth(),pos.y,headsizex-headedgewidth(),size.y-1);
      drawOutlineRect();
    }
    else
    {
      fill(colorhue,1,opacityleft);//LEFT HIGHLIGHTED
      rect(pos.x,pos.y,headedgewidth(),headsizey);
      beginShape();
      vertex(pos.x,pos.y+headsizey);
      vertex(pos.x+headedgewidth(),pos.y+headsizey);
      vertex(pos.x+min(size.x,headedgewidth()),pos.y+size.y);
      vertex(pos.x,pos.y+size.y);
      endShape();
      fill(colorhue,1,opacitycenter);//RIGHT DARKER
      rect(pos.x+headedgewidth(),pos.y,headsizex-headedgewidth(),headsizey);
      beginShape();
      vertex(pos.x+headedgewidth(),pos.y+headsizey);
      vertex(pos.x+headsizex,pos.y+headsizey);
      vertex(pos.x+size.x,pos.y+size.y);
      vertex(pos.x+min(size.x,headedgewidth()),pos.y+size.y);
      endShape();
      drawOutlinePolygon();
    }
  }
  else if((mouseOverHeadRight() && gui.canvas.dragmode<0) || (gui.canvas.dragmode==2 && dragging))
  {
    if(size.x==headsizex)
    {
      fill(colorhue,1,opacitycenter);
      rect(pos.x,pos.y,size.x-headedgewidth(),size.y-1);
      fill(colorhue,1,opacityright);
      rect(pos.x+size.x-headedgewidth()+1,pos.y,headedgewidth()-1,size.y-1);
      drawOutlineRect();
    }
    else
    {
      fill(colorhue,1,opacitycenter);//LEFT DARKER
      rect(pos.x,pos.y,headsizex-headedgewidth(),headsizey);
      beginShape();
      vertex(pos.x,pos.y+headsizey);
      vertex(pos.x+headsizex-headedgewidth(),pos.y+headsizey);
      vertex(pos.x+min(size.x,headsizex-headedgewidth()),pos.y+size.y);
      vertex(pos.x,pos.y+size.y);
      endShape();
      fill(colorhue,1,opacityright);//RIGHT HIGHLIGHTED
      rect(pos.x+headsizex-headedgewidth(),pos.y,headedgewidth(),headsizey);
      beginShape();
      vertex(pos.x+headsizex-headedgewidth(),pos.y+headsizey);
      vertex(pos.x+headsizex,pos.y+headsizey);
      vertex(pos.x+size.x,pos.y+size.y);
      vertex(pos.x+min(size.x,headsizex-headedgewidth()),pos.y+size.y);
      endShape();
      drawOutlinePolygon();
    }
  }
  else
  {
    fill(colorhue,1,opacitycenter);
    if(size.x>=headsizex)
    {
```

```
        rect(pos.x,pos.y,size.x,size.y-1);
        drawOutlineRect();
      }
      else
      {
        rect(pos.x,pos.y, headsizex,headsizey);
        beginShape();
        vertex(pos.x,pos.y+headsizey);
        vertex(pos.x+headsizex,pos.y+headsizey);
        vertex(pos.x+size.x,pos.y+size.y);
        vertex(pos.x,pos.y+size.y);
        endShape();
        drawOutlinePolygon();
      }
    }
  }
  void displayHeadText()
  {
    //text: name + id
    textFont(fontsmall);
    String name = region.name();
    float namewidth=textWidth(name);
    float idwidth=textWidth(""+region.id);
    if(namewidth>headsizex)
      name=name.substring(0, int(headsizex / namewidth * name.length()));
    fill(1);
    text(name,pos.x+5,pos.y+10);
    if(namewidth+textWidth(""+region.id)+10<headsizex)
      text(""+region.id, pos.x+headsizex-idwidth-5,pos.y+10);
    //the thin line below the header
    stroke(0,0,1,.3);
    line(pos.x,pos.y+headsizey-1,pos.x+headsizex,pos.y+headsizey-1);
    noFill();
    stroke(0);
    if(headsizex==size.x)
      rect(pos.x,pos.y,headsizex,size.y);
    if(selected)
    {
      noFill();
      stroke(0,0,1,.8);
      rect(pos.x+1,pos.y,headsizex-1,size.y-1);
      //    stroke(0,0,millis()*.011%1,.8);
      rect(pos.x,pos.y-1,headsizex+1,size.y+1);
    }
  }
  final void drawOutlinePolygon()
  {
    noFill();
    stroke(0);
    beginShape();
    vertex(pos.x,pos.y);
    vertex(pos.x+headsizex,pos.y);
    vertex(pos.x+headsizex,pos.y+headsizey);
    vertex(pos.x+size.x,pos.y+size.y);
    vertex(pos.x,pos.y+size.y);
    vertex(pos.x,pos.y);
    endShape();
  }
  final void drawOutlineRect()
  {
    noFill();
    stroke(0);
    rect(pos.x,pos.y,size.x,size.y);
  }
  final float headedgewidth()
  {
    return constrain(headsizex*.1,6,50);
  }
  final boolean mouseOverHead()
  {
    return mouseY>=pos.y && mouseY<pos.y+headsizey && mouseX>=pos.x && mouseX<pos.x+headsizex;
  }
  final boolean mouseOverHeadCenter()
  {
    return mouseY>=pos.y && mouseY<pos.y+headsizey && mouseX>=pos.x+headedgewidth() && mouseX<pos.x+headsizex-headedgewidth();
  }
  final boolean mouseOverHeadLeft()
  {
    return mouseY>=pos.y && mouseY<pos.y+headsizey && mouseX>=pos.x && mouseX<pos.x+headedgewidth();
  }
  final boolean mouseOverHeadRight()
  {
    return mouseY>=pos.y && mouseY<pos.y+headsizey && mouseX>=pos.x+headsizex-headedgewidth() && mouseX<pos.x+headsizex;
  }
  final boolean mouseOverHeadSide()
  {
    return mouseY>=pos.y && mouseY<pos.y+headsizey && ((mouseX>=pos.x && mouseX<pos.x+headedgewidth()) || (mouseX>=pos.x+headsizex-headedgewidth() && m
  }
  final boolean mouseOverBody()
  {
    return mouseY>pos.y+headsizey && mouseY<pos.y+size.y && mouseX>=pos.x && mouseX<pos.x+size.x;
  }
};

public class RegionWithChildrenView extends RegionView{
  final private RegionWithChildren regionwithchildren; //to avoid typecasting all the time - any better ideas?
  private ArrayList rows;
  RegionWithChildrenView(RegionWithChildren _region)
  {
    region=_region;
```

```
  regionwithchildren=(RegionWithChildren)_region;
  pos=new Coord(0,0);
  size=new Coord(0,0);
  rows = new ArrayList();
}
void updateGeneration(int _generation)
{
  generation=_generation;
  for(int ichild=0;ichild<regionwithchildren.countChildren();ichild++)
    regionwithchildren.getChild(ichild).view.updateGeneration(generation+1);
}
float zoomY()
{
  return max(1,sqrt(rows.size()));
}
void putChildViewInRow(RegionView childview)
{
  for(int irow=0;irow<rows.size();irow++)
  {
    RegionWithChildrenViewRow row = (RegionWithChildrenViewRow) rows.get(irow);
    if(row.tryToFitChildViewIn(childview))//check for similar zoomy and overlaps on x-axis
      return;
  }
  rows.add(new RegionWithChildrenViewRow(childview));
}
void updateXYDownTree(float posy, float sizey)
{
  updateX();
  pos.y=posy;
  size.y=sizey;
  int nchildren = regionwithchildren.countChildren();
  //create rows
  rows = new ArrayList();
  for(int ichild=0;ichild<nchildren;ichild++)
    putChildViewInRow(regionwithchildren.getChild(ichild).view);
  //sum up zoomY
  float rowszoomysum = 0;
  for(int irow=0;irow<rows.size();irow++)
  {
    RegionWithChildrenViewRow row = (RegionWithChildrenViewRow) rows.get(irow);
    rowszoomysum+=row.zoomy;
  }
  //set pos.y and size.y
  posy = pos.y+headsizey;
  for(int irow=0;irow<rows.size();irow++)
  {
    RegionWithChildrenViewRow row = (RegionWithChildrenViewRow) rows.get(irow);
    sizey = (size.y-headsizey) * row.zoomy / rowszoomysum;
    row.setChildViewsY(posy,sizey);
    posy += sizey;
  }
}
void run()
{
  actionmenu();
  if(region==null)
    return;
  move();
  displayBody();
  displayFades();
  displayHeadText();
  for(int ichild=0;ichild<regionwithchildren.countChildren();ichild++)
    regionwithchildren.getChild(ichild).view.run();
}
void displayFades()
{
  noStroke();
  fill(0,0,0,.5);
    float xrange,xinc;
  //fade in
  if(regionwithchildren.fade.intime.getFloat()>0)
  {
    beginShape();
    vertex(pos.x,pos.y);
    vertex(pos.x,pos.y+headsizey);
    xrange = max(0,regionwithchildren.fade.intime.getFloat() * gui.canvas.pixelspersecond);
    xinc=sqrt(xrange);
    for(float x=pos.x; x<pos.x+size.x && x<pos.x+xrange; x+=xinc)
      vertex(x, pos.y+headsizey-2 - (headsizey-3)*pow((x-pos.x)/xrange, regionwithchildren.fade.incurve.getFloat()));
    vertex(pos.x+min(size.x,xrange),pos.y);
    endShape();
  }
  //fade out
  if(regionwithchildren.fade.outtime.getFloat()>0)
  {
    beginShape();
    vertex(pos.x+size.x,pos.y);
    vertex(pos.x+size.x,pos.y+headsizey);
    xrange = max(0,regionwithchildren.fade.outtime.getFloat() * gui.canvas.pixelspersecond);
    xinc=-sqrt(xrange);
    for(float x=pos.x+size.x; x>=pos.x && x>pos.x+size.x-xrange; x+=xinc)
      vertex(x, pos.y+headsizey-2 - (headsizey-3)*pow((pos.x+size.x-x)/xrange, regionwithchildren.fade.outcurve.getFloat()));
    vertex(pos.x+size.x-min(size.x,xrange),pos.y);
    endShape();
  }
}
int countSelectedChildren()
{
  ArrayList affe=new ArrayList();
```

```
    collectSelectedChildren(affe);
    return affe.size();
  }
  void  collectSelectedChildren(ArrayList result)
  {
    for(int ichild=0;ichild<regionwithchildren.countChildren();ichild++)
      if(regionwithchildren.getChild(ichild).view.selected)
        result.add(regionwithchildren.getChild(ichild));
  }
  void collectSelectedAncestors(ArrayList result)
  {
    for(int ichild=0;ichild<regionwithchildren.countChildren();ichild++)
    {
      Region child=regionwithchildren.getChild(ichild);
      if(child.view.selected)
        result.add(child);
      if(!child.isEnvRegion)
        ((RegionWithChildrenView)child.view).collectSelectedAncestors(result);
    }
  }
  void unselectAncestors()
  {
    selected=false;
    for(int ichild=0;ichild<regionwithchildren.countChildren();ichild++)
      regionwithchildren.getChild(ichild).view.unselectAncestors();
  }
};


public class EnvRegionView extends RegionView{
  final private EnvRegion envregion; //to avoid typecasting all the time - any better ideas?
  float minimum, maximum;
  float heavyminimum, heavymaximum;
  EnvRegionView(EnvRegion _region)
  {
    region=_region;
    envregion=(EnvRegion)_region;
    pos=new Coord(0,0);
    size=new Coord(0,0);
  }
  float zoomY()
  {
    float result = 1;
    //  if(gui.inspector.region==region)
    //    result*=2;
    return result;
  }
  void updateBreakpointsMinimumAndMaximum()
  {
    if(envregion.countBreakpoints()<1)
      return;
    minimum = 0;
    maximum = 1;
    heavyminimum = heavymaximum = envregion.getBreakpoint(0).getHeavyValueAt(1);
    float y, yheavy;
    for(int ibreakpoint=0;ibreakpoint<envregion.countBreakpoints();ibreakpoint++)
    {
      y=envregion.getBreakpoint(ibreakpoint).lightValueAt(1);
      yheavy=envregion.getBreakpoint(ibreakpoint).getHeavyValueAt(1);
      if(y<minimum)
        minimum=y;
      if(y>maximum)
        maximum=y;
      if(yheavy<heavyminimum)
        heavyminimum=yheavy;
      if(yheavy>heavymaximum)
        heavymaximum=yheavy;
    }
  }
  void updateXYDownTree(float posy, float sizey)
  {
    updateX();
    pos.y=posy;
    size.y=sizey;
    updateBreakpointsMinimumAndMaximum();
  }
  void run()
  {
    actionmenu();
    move();
    displayBody();
    displayHeadText();
    editBreakpoints();
    displayBreakpoints();
  }
  void editBreakpoints()
  {
    if(region!=gui.inspector.selectedenvregion)
    {
      return;
    }
    if(clickpending==LEFT && mouseOverBody()) //NEW BREAKPOINT?
    {
      if(keycodesdown[SHIFT])
      {
        envregion.newBreakpoint(gui.canvas.pos2seconds(mouseX), 1-(mouseY-pos.y-headsizey)/(size.y-headsizey));
        clickpending=0;
      }
    }
    pushMatrix();
    translate(0,0,2);
```

```
  boolean xoutofrange=false;
  for(int ibreakpoint=0;ibreakpoint<envregion.countBreakpoints() &! xoutofrange;ibreakpoint++)
  {
    Breakpoint e = envregion.getBreakpoint(ibreakpoint);
    float heavyx = gui.canvas.seconds2pos(e.heavyStartTime());
    if(heavyx>gui.canvas.pos.x+gui.canvas.size.x)xoutofrange=true;
    if(!xoutofrange)
    {
      float heavyy = pos.y+headsizey +1+ (1-e.getHeavyValueAt(1))*(size.y-headsizey-4);
      if(mouseX>heavyx-4 && mouseX<heavyx+4 && mouseY>heavyy-4 && mouseY<heavyy+4)
      {
        if(clickpending==LEFT)
        {
          clickpending=0;
          gui.canvas.draggingbreakpoint=e;
        }
        stroke(0);
        rect(heavyx-3, heavyy-3, 7,7);
      }
      else
      {
        noStroke();
        fill(1);
        rect(heavyx-1, heavyy-1, 3,3);
      }
      if(e==gui.canvas.draggingbreakpoint)
      {
        heavyx=mouseX;// constrain(mouseX, pos.x,pos.x+size.x);
        heavyy=constrain(mouseY, pos.y+headsizey,pos.y+size.y);
        e.setHeavyStartTime(gui.canvas.pos2seconds(heavyx));
        float yval = 1-(heavyy-pos.y-headsizey)/(size.y-headsizey);
        e.setHeavyValueAt(1, yval);
        updateBreakpointsMinimumAndMaximum();
        if(mousePressed==false)//drop
        {
          displayneedsupdate = envregion.sortBreakpointsByTime();
          gui.canvas.draggingbreakpoint=null;
          session.markDirty();
        }
        //text x+y values next to mouse pointer
        if(e==gui.canvas.draggingbreakpoint || mouseX>heavyx-4 && mouseX<heavyx+4 && mouseY>heavyy-4 && mouseY<heavyy+4)
        {
          float yvalposy=mouseY-7;
          if(heavyy>e.lightValueAt(1)+5)
            yvalposy+=35;
          if(yvalposy>height-25)
            yvalposy-=60;
          fill(1);
        }
      }
    }
    //remove breakpoint?
    if(e==gui.canvas.draggingbreakpoint && (keypending==BACKSPACE || keypending==DELETE))
    {
      keypending=0;
      gui.canvas.draggingbreakpoint=null;
      envregion.removeBreakpoint(ibreakpoint);
      ibreakpoint--;
      envregion.sortBreakpointsByTime();
      displayneedsupdate = true;
      updateBreakpointsMinimumAndMaximum();
      session.markDirty();
    }
  }
  popMatrix();
}
void displayBreakpoints()
{
  if(!transport.playing && gui.canvas.draggingbreakpoint==null && frameCount%50==region.id%50)
    updateBreakpointsMinimumAndMaximum();
  pushMatrix();
  translate(0,0,1);
  if(region==gui.inspector.selectedenvregion)
  {
    fill(1);
    text("maximum: "+maximum, pos.x+3,pos.y+headsizey+13);
    text("minimum: "+minimum, pos.x+3,pos.y+size.y-3);
  }
  float lightx,lighty;
  float prevlightx=0,prevlighty=0;
  for(int ibreakpoint=0;ibreakpoint<envregion.countBreakpoints();ibreakpoint++)
  {
    Breakpoint e = envregion.getBreakpoint(ibreakpoint);
    lightx = gui.canvas.seconds2pos(e.lightStartTime());
    lighty = pos.y+size.y -4- ( e.lightValueAt(1)-minimum)/(maximum-minimum)  * (size.y-headsizey-4);
    if(region==gui.inspector.selectedenvregion)
    {
      //black lines from heavy to light
      stroke(0,0,0,.5);
      float heavyx = gui.canvas.seconds2pos(e.heavyStartTime());
      float heavyy = pos.y+size.y -4- e.getHeavyValueAt(1) * (size.y-headsizey-4);
      if(heavyx>pos.x && heavyx<pos.x+size.x)
      {
        line(heavyx,heavyy,lightx,lighty);
        line(heavyx+1,heavyy,lightx+1,lighty);
        line(heavyx,heavyy+1,lightx,lighty+1);
      }
    }
    if(ibreakpoint>0)
```

```
      {
        if(region==gui.inspector.selectedenvregion)
        {
          stroke(0,0,1);
          drawCurve(prevlightx,prevlighty+1,lightx, lighty+1, pos.x, pos.x+size.x, envregion.interpolation.getFloat(), 8);
        }
        stroke((0000 *.3)%1,.3,1, .3);
        drawCurve(prevlightx,prevlighty,lightx, lighty, pos.x, pos.x+size.x, envregion.interpolation.getFloat(), 8);
      }
      prevlightx=lightx;
      prevlighty=lighty;
    }
    popMatrix();
  }
};

public class RegionWithChildrenViewRow{
  float zoomy;
  ArrayList childviews;
  RegionWithChildrenViewRow(RegionView childview)
  {
    zoomy = childview.zoomY();
    childviews = new ArrayList();
    childviews.add(childview);
  }
  boolean tryToFitChildViewIn(RegionView childview)
  {
    if(zoomy<childview.zoomY()*.5 || zoomy>childview.zoomY()*2)
      return false;
    boolean fits=true;
    for(int ichildview=0; ichildview<childviews.size() && fits; ichildview++)
    {
      RegionView cv = (RegionView)childviews.get(ichildview);
      //overlap on x axis?
      if((cv.pos.x >= childview.pos.x && cv.pos.x < childview.pos.x+childview.size.x) ||
        (cv.pos.x+cv.size.x >= childview.pos.x && cv.pos.x+cv.size.x < childview.pos.x+childview.size.x) ||
        (childview.pos.x >= cv.pos.x && childview.pos.x < cv.pos.x+cv.size.x))
        fits=false;
    }
    if(fits)
      childviews.add(childview);
    return fits;
  }
  void setChildViewsY(float posy, float sizey)
  {
    for(int ichildview=0; ichildview<childviews.size(); ichildview++)
      ((RegionView)childviews.get(ichildview)).updateXYDownTree(posy,sizey-4);
  }
};
```

rohr.pde

```
import oscP5.*;
import netP5.*;
OscP5 oscP5;

public class Rohr{
  NetAddress myRemoteLocation;
  boolean session_connected;
  Rohr()
  {
    /* start oscP5, listening for incoming messages at port 17000 */
    oscP5 = new OscP5(this,17000);
    /* sc is listening at port 57120 */
    myRemoteLocation = new NetAddress("127.0.0.1",57120);
    println("********************************");
  }
  void oscEvent(OscMessage msg)
  {
    displayneedsupdate=true;
  //   println("OSC received");
    if(identify(msg,"/connect_session",""))
    {
      debug("SESSION CONNECTED TO SC");
      session_connected=true;
      return;
    }
    if(identify(msg,"/transport_updateplaypos","f"))
    {
      transport.remoteUpdatePlayPos(msg.get(0).floatValue());
      return;
    }
    if(identify(msg,"/transport_locate","f"))
    {
      transport.remoteLocate(msg.get(0).floatValue());
      return;
    }
    if(identify(msg,"/transport_play", ""))
    {
```

```
    transport.remotePlay();
    return;
  }
  if(identify(msg,"/transport_stop", ""))
  {
    transport.remoteStop();
    return;
  }
  if(identify(msg,"/recorded_midi_breakpoint", "ff"))
  {
    transport.tempmidiinput.add(new Coord(msg.get(0).floatValue(), msg.get(1).floatValue()));
    return;
  }
  if(identify(msg,"/new_buffer", "s"))
  {
    session.newBuffer(msg.get(0).stringValue());
    return;
  }
  warn("unknown OSC message with typetag "+msg.typetag());
}
void send(String command)
{
  println("sending OSC: "+command);
  OscMessage msg = new OscMessage(command);
  oscP5.send(msg, myRemoteLocation);
}
void send(String command, float arg)
{
  println("sending OSC: "+command+" "+arg);
  OscMessage msg = new OscMessage(command);
  msg.add(arg);
  oscP5.send(msg, myRemoteLocation);
}
void send(String command, boolean arg)// sc empfaengt bool-werte als falsche floats, darum 1/0 statt true/false
{
  println("sending OSC: "+command+" "+arg);
  OscMessage msg = new OscMessage(command);
  msg.add(arg ? 1 : 0);
  oscP5.send(msg, myRemoteLocation);
}
void send(String command, String arg)
{
  println("sending OSC: "+command+" "+arg);
  OscMessage msg = new OscMessage(command);
  msg.add(arg);
  oscP5.send(msg, myRemoteLocation);
}
void send(String command, float arg1, float arg2)
{
  println("sending OSC: "+command+" "+arg1+"   "+arg2);
  OscMessage msg = new OscMessage(command);
  msg.add(arg1);
  msg.add(arg2);
  oscP5.send(msg, myRemoteLocation);
}
void send(String command, String arg1, float arg2)
{
  println("sending OSC: "+command+" "+arg1+"   "+arg2);
  OscMessage msg = new OscMessage(command);
  msg.add(arg1);
  msg.add(arg2);
  oscP5.send(msg, myRemoteLocation);
}
void send(String command, String arg1, float arg2, float arg3)
{
  println("sending OSC: "+command+" "+arg1+"   "+arg2+"   "+arg3);
  OscMessage msg = new OscMessage(command);
  msg.add(arg1);
  msg.add(arg2);
  msg.add(arg3);
  oscP5.send(msg, myRemoteLocation);
}
void send(String command, String arg1, float arg2, float arg3, float arg4)
{
  println("sending OSC: "+command+" "+arg1+"   "+arg2+"   "+arg3+"   "+arg4);
  OscMessage msg = new OscMessage(command);
  msg.add(arg1);
  msg.add(arg2);
  msg.add(arg3);
  msg.add(arg4);
  oscP5.send(msg, myRemoteLocation);
}
void send(String command, String arg1, float arg2, float arg3, float arg4, float arg5)
{
  println("sending OSC: "+command+" "+arg1+"   "+arg2+"   "+arg3+"   "+arg4+"   "+arg5);
  OscMessage msg = new OscMessage(command);
  msg.add(arg1);
  msg.add(arg2);
  msg.add(arg3);
  msg.add(arg4);
  msg.add(arg5);
  oscP5.send(msg, myRemoteLocation);
}
boolean identify(OscMessage msg, String command, String typetag)
{
  if(!msg.checkAddrPattern(command))
    return false;
  if(msg.checkTypetag(typetag))
    return true;
```

```
    warn("OSC message "+command+" received with wrong type tag "+msg.typetag()+"\nmust be "+typetag);
    return false;
  }
};
 void oscEvent(OscMessage msg)
 {
  fail("fa");
 }
```

session.pde

```
public class Session{
  public String loadedpath, name, projectfolder, undopath, redopath;
  RegionWithChildren rootregion;
  ArrayList buffers;
  int versionnumber, filecounter;
    private int IDoffset;//no free ids below this number
  private boolean dirty=false;
  public Session()
  {
    loadedpath="--new session--";
    int randomnumber=int(random(10000,100000));
    name="ysession"+randomnumber;
    projectfolder="/home/stefan/Y/"+name;
    undopath="";
    redopath="";
    versionnumber=1;
    buffers=new ArrayList();
    rootregion=new GroupRegion(null, "root", 0, makeFloatInputFieldTargetArray(0,4));
    rootregion.init();
    dirty=false;
  }
  void run()
  {
    if(dirty)
      save();
  }
  void markDirty()
  {
//    debug("markDirty");
    dirty=true;
  }
  String currentPath()
  {
    return projectfolder+"/"+name+".xml";
  }
  RegionWithChildren getRootRegion()
  {
    return rootregion;
  }
  Region getRegionByID(int the_id)
  {
    if(the_id<0)
    {
      warn("you asked session.getRegionByID for a negative ID "+the_id+"\nmaybe you wanted to call region.getRegionByID");
    }
    return rootregion.getDescendantByID(the_id);
  }
  Region getRegionByName(String the_name)
  {
    return rootregion.getDescendantByName(the_name);
  }
  void checkRouting(Region r)
  {
    println("TODO: checkrouting");
  }
  void updateGenerations()
  {
    rootregion.view.updateGeneration(0);
  }
  float startTime()
  {
    return rootregion.lightStartTime();
  }
  float endTime()
  {
    return rootregion.lightEndTime();
  }
  void save()
  {
    if(!dirty)
      return;
    save(currentPath());
  }
  void saveForced()
  {
    save(currentPath());
  }
  void save(String path)
```

```
  {
    sessionmanager.blocked=true;
    undopath = projectfolder+"/backup/"+name+"_"+versionnumber+".xml";
    moveFile(projectfolder+"/"+name+".xml", undopath);
    redopath="";
    versionnumber++;
    debug(path);
    xmlInOut.saveElement(getXMLElement(), path);
    println("session saved: "+path);
    dirty=false;
    sessionmanager.blocked=false;
  }
  void initForGUI()
  {
    println("initializing session for GUI...");
    updateGenerations();
    rootregion.view.updateGeneration(0);
    checkRouting(rootregion);
    gui.inspector.updateViews(rootregion);
    gui.canvas.focusRegionView(rootregion.view);
    gui.canvas.updateXY();
    gui.masterview.sessionpathview.pathtarget.setStringAndNothingElse(loadedpath);
    dirty=false;
    sessionmanager.blocked=false;
    println("... session ready");
  }
  int nextFreeID()
  {
    while(getRegionByID(IDoffset)!=null)
      IDoffset++;
    return IDoffset;
  }
  XMLElement getXMLElement()
  {
    XMLElement result=new XMLElement("session");
    result.addAttribute("name",name);
    result.addAttribute("versionnumber",versionnumber);
    result.addAttribute("projectfolder",projectfolder);
    result.addAttribute("undopath",undopath);
    result.addAttribute("redopath",redopath);
    result.addAttribute("filecounter",filecounter);
    result.addAttribute("IDoffset",IDoffset);
    result.addChild(rootregion.getXMLElement());
    for(int i=0; i<countBuffers(); i++)
      result.addChild(getBuffer(i).getXMLElement());
    return result;
  }
  void destroy()
  {
    println("TODO: destroysession");
  }
  int countBuffers()
  {
    return buffers.size();
  }
  Buffer getBuffer(int index)
  {
    return (Buffer) buffers.get(index);
  }
  void addBuffer(Buffer buffer)
  {
    buffers.add(buffer);
  }
  void removeBuffer(int index)
  {
    buffers.remove(index);
  }
  void removeAllBuffers()
  {
    buffers=new ArrayList();
  }
  void newBuffer(String path)
  {
   addBuffer(new Buffer(path, smallestFreeBufnum(0)));
  }
  void listBuffers()
  {
    println("____________________ BUFFERS");
    for(int i=0; i<countBuffers(); i++)
      println(getBuffer(i).bufnum+"      "+getBuffer(i).path);
  }
  int smallestFreeBufnum(int start_bufnum)
  {
    for(int i=0; i<countBuffers(); i++)
      if(getBuffer(i).bufnum==start_bufnum)
        return smallestFreeBufnum(start_bufnum+1);
    return start_bufnum;
  }
  int largestUsedBufnum()
  {
    int result=0;
    for(int i=0; i<countBuffers(); i++)
      if(getBuffer(i).bufnum>result)
        result=getBuffer(i).bufnum;
    return result;
  }
  ////////////////////////////////// SCRIPTING ONLY
  String getProjectFolder()
  {
  return projectfolder;
}
String getTempFilePath(String prefix)
```

```
{
    filecounter++;
    String numberstring=""+filecounter;
    while(numberstring.length()<6)
    numberstring="0"+numberstring;
  return projectfolder+"/"+prefix+numberstring+"_"+versionnumber+".wav";
}
};
```

sessionmanager.pde

```
public class SessionManager{
  boolean blocked=true;
  String pathToLoad;
  SessionManager()
  {
    xmlInOut = new XMLInOut(the_applet);
    session=new Session();
  }
  void load(String path)
  {
    loadOnly(path);
  }
  void loadOnly(String path)
  {
    sessionmanager.blocked=true;
    try{
      xmlInOut.loadElement(path);
    }
    catch(Exception e){
      warn("could not load xml file "+path);
      return;
    }
    pathToLoad=path;
    debug("pflanzen");
    wait_until_unblocked();
    debug("fresser");
    connect_session_to_sc();
  }
  void undo()
  {
    debug("UNDO " +session.versionnumber);
    if(session.undopath.length()<1)
    {
      warn("can't undo");
      return;
    }
    //save this version
    xmlInOut.saveElement(session.getXMLElement(), session.projectfolder+"/backup/"+session.name+"_"+session.versionnumber+".xml");
    //load older version from undopath
    moveFile(session.undopath, session.currentPath());
    load(session.currentPath());
    //session points to the new session if successful
    session.redopath = session.projectfolder+"/backup/"+session.name+"_"+(session.versionnumber+1)+".xml";
    gui.canvas.updateAfterUndo();
    gui.inspector.updateViews(session.getRegionByID(gui.inspector.region.id));
  }
  void redo()
  {
    debug("REDO " +session.versionnumber);
    if(session.redopath.length()<1)
    {
      warn("can't redo");
      return;
    }
    //load version from redopath
    moveFile(session.currentPath(), session.projectfolder+"/backup/"+session.name+"_"+session.versionnumber+".xml");
    moveFile(session.redopath, session.currentPath());
    load(session.currentPath());
    //session points to the new session if successful
    gui.canvas.updateAfterUndo();
    gui.inspector.updateViews(session.getRegionByID(gui.inspector.region.id));
  }
  void reload()
  {
    debug("RELOAD "+session.versionnumber);
    load(session.currentPath());
    //session points to the new session if successful
    gui.canvas.updateAfterUndo();
    gui.inspector.updateViews(session.getRegionByID(gui.inspector.region.id));
  }
  void parseSession(XMLElement element)
  {
    wait_for_synthdatabase();
    Session newsession = new Session();
    try{
      newsession.name = element.getAttribute("name");
```

```
      newsession.versionnumber = element.getIntAttribute("versionnumber");
      newsession.projectfolder = element.getAttribute("projectfolder");
      newsession.undopath = element.getAttribute("undopath");
      newsession.redopath = element.getAttribute("redopath");
      newsession.filecounter = element.getIntAttribute("filecounter");
      newsession.IDoffset = element.getIntAttribute("IDoffset");
    }
    catch(InvalidAttributeException e)
    {
      warn("error in XML file: session attributes missing");
      return;
    }
    XMLElement root = element.getChild(0);
    if(!root.getName().equals("groupregion"))
    {
      warn("error in XML file: root region must be a group");
      return;
    }
    newsession.rootregion=parseGroupRegion(element.getChild(0),null);
    if(newsession.rootregion==null)
    {
      warn("error in XML file");
      return;
    }
    //parse buffers
    for(int i = 1; i < element.countChildren();i++)
    {
      XMLElement child = element.getChild(i);
      if(!child.getName().equals("buffer"))
      {
        warn("illegal xml element at top level: "+child.getName()+"  expecting buffer");
        return;
      }
      newsession.addBuffer(new Buffer(child.getAttribute("path"), child.getIntAttribute("bufnum")));
    }
    newsession.loadedpath=pathToLoad;
    debug("session.destroy();");
    session.destroy();
    session=newsession;
    debug("session.initForGUI();");
    session.initForGUI();
    println("session parsed: "+session.name);
  }
  void wait_until_unblocked()
  {
    for(int milliseconds=1; blocked && milliseconds<7000 ; milliseconds*=2)
    try {
      Thread.sleep(milliseconds);
    }
    catch (InterruptedException e) {
    }
  }
  void connect_session_to_sc()
  {
    rohr.session_connected=false;
    rohr.send("/connect_session");
    transport.locate(transport.curtime);//this sends cursor time to sc
    for(int milliseconds=1; (rohr.session_connected==false || oscP5==null) && milliseconds<2000 ; milliseconds*=2)
    try {
      print(".");
      Thread.sleep(milliseconds);
    }
    catch (InterruptedException e) {
    }
    if(!rohr.session_connected)
    {
      warn("could not connect session to supercollider");
      return;
    }
  }
};
```

swingosc.pde

```
SwingOSCThread swingoscthread;
public class SwingOSCThread extends Thread{
  void run()
  {
    SwingOSC.main(new String[]{"-t", "57111", "-L", "-h", "127.0.0.1:57120"});
  }
};

void setupSwingOSC()
{
  SwingOSCPetze.applet=the_applet;
  swingoscthread = new SwingOSCThread();
  swingoscthread.start();
}
```

```
public Session getSession()
{
  return session;
}
public SynthDatabase getSynthDatabase()
{
  return synthdatabase;
}
public SessionManager getSessionManager()
{
  return sessionmanager;
}
////////////////////////////////////////////////// important for SwingOSC to build the Synthdatabase here
public Synth newSynth(String name)
{
  return new Synth(name);
}
public AudioParam newAudioParam(String name, float value, boolean isrouting)
{
  return new AudioParam(name, value, isrouting);
}
public NumberParam newNumberParam(String name, float value)
{
  return new NumberParam(name, value);
}
```

synth.pde

```
public class Synth{
  String name;
  SynthRegion region;
  ArrayList params;
  public Synth(String _name)
  {
    name=_name;
    params=new ArrayList();
  }
  int countParams()
  {
    return params.size();
  }
  Param getParam(int index)
  {
    return (Param) params.get(index);
  }
  void addParam(Param p)
  {
    p.synth=this;
    params.add(p);
  }
    XMLElement getXMLElement()
  {
    XMLElement result=new XMLElement("synth");
    result.addAttribute("name",name);
    for(int iparam=0;iparam<params.size();iparam++)
      result.addChild(getParam(iparam).getXMLElement());
    return result;
  }
  Synth copy()
  {
    Synth newSynth = new Synth(name);
    for(int iparam=0;iparam<params.size();iparam++)
      newSynth.addParam(((Param)getParam(iparam)).copy());
    return newSynth;
  }
  ////////////////////////// SCRIPTING ONLY
  String getName()
  {
    return name;
  }
  void setParam(int index, float value)
  {
    getParam(index).setString(""+value);
  }
  void setParam(int index, String value)
  {
    getParam(index).setString(""+value);
  }
};

public class SynthDatabase{
  boolean ready;
  ArrayList synths;
  SynthDatabase()
  {
    synths = new ArrayList();
```

```
  }
  void init()
  {
    rohr.send("/transmit_synthdatabase");
  }
  void addSynth(Synth s)
  {
    synths.add(s);
  }
  Synth create(String name)
  {
    Synth newSynth = getSynthByName(name);
    if(newSynth==null)
    {
      warn("synth '"+name+"' not found");
      return null;
    }
    return newSynth.copy();
  }
  String[] getSynthNames()
  {
   String[] result=new String[synths.size()];
    for(int i=0; i<synths.size(); i++)
    result[i]=getSynth(i).name;
    return result;
  }
  int getIndexOf(String name)
  {
    for(int i=0; i<synths.size(); i++)
      if(name.equals(((Synth)synths.get(i)).name))
        return i;
    return -1;
      }
  Synth getSynthByName(String name)
  {
    for(int i=0; i<synths.size(); i++)
      if(name.equals(((Synth)synths.get(i)).name))
        return (Synth)synths.get(i);
    return null;
  }
  Synth getSynth(int index)
  {
    return (Synth)synths.get(index);
  }
  void remoteTransmitDone()
  {
    ready=true;
  }
  void print()
  {
    println("_____________________________________________SynthDatabase");
    for(int i=0; i<synths.size(); i++)
    {
      Synth synth= getSynth(i);
      println(">>>>>>>>>>>>>>>>>>>>>  "+synth.name);
      for(int iparam=0; iparam<synth.countParams();iparam++)
        synth.getParam(iparam).print();
    }
  }
};
```

timelineview.pde

```
public class TimeLineView extends GUIElement{
  float starttime, endtime, mousetime;
  Coord pos, size;
  float dragstarttime, dragstartpos, mousepos;
  boolean dragging=false;
  TimeLineView(Coord _pos, Coord _size)
  {
    pos=_pos;
    size=_size;
  }
  void run()
  {
    pushMatrix();
    translate(0,0,2);
    starttime=gui.canvas.pos2seconds(pos.x);
    endtime=gui.canvas.pos2seconds(pos.x+size.x);
    mousetime=gui.canvas.pos2seconds(mouseX);
    mousepos=mouseX;
    noStroke();
    fill(1,1,1,.3);
    rect(pos.x,pos.y,size.x,size.y);
    fill(0,1,1);
    stroke(0,1,0);
    textFont(fontsmall);

    int maxvalues=10;
```

```
    float rawchroma=(endtime-starttime) / maxvalues;
    float chroma=pow(10,floor(log10(rawchroma))); // 10 | 1 | .1 | .01 | .001 ...
    if((endtime-starttime)/chroma > maxvalues*5)
      chroma/=.2;
    if((endtime-starttime)/chroma > maxvalues*2)
      chroma/=.5;
    displayExactTimePoint(starttime,.01);
    float t;
    for(int i=1; (t=starttime+i*chroma)<endtime-chroma*.3; i++)
      displayRoundedTimePoint(t,chroma);
    boolean mouseOver = mousepos>pos.x && mousepos<pos.x+size.x && mouseY>pos.y && mouseY<pos.y+size.y;
    if(mouseOver)
    {
      gui.canvas.lineVertical(mousepos, color(.6,1,.7));
    }
    if(dragging)
    {
      if(mousePressed)
      {
        keypending=0;
      }
      else
      {
        dragRelease();
      }
    }
    else //not dragging
    {
      if(mouseOver && clickpending==LEFT)
      {
        dragging=true;
        dragstarttime=mousetime;
        dragstartpos=mousepos;
        clickpending=0;
        displayneedsupdate=true;
      }
    }
    popMatrix();
  }
  void dragRelease()
  {
    dragging=false;
    transport.loopstarttime=min(dragstarttime, mousetime);
    transport.loopendtime=max(dragstarttime, mousetime);
    transport.looping=dragstartpos!=mousepos;
    if(transport.looping)
      println("loop "+transport.loopstarttime+"   "+transport.loopendtime);
    transport.locate(transport.loopstarttime);
    clickpending=0;
    displayneedsupdate=true;
  }
  void display()
  {
    if(dragging)
      gui.canvas.rectVertical(dragstartpos, mousepos, color(.6,.7,1,.7), color(.6,.7,1,.3));
    if(transport.looping &! dragging)
      gui.canvas.rectVertical(gui.canvas.seconds2pos(transport.loopstarttime), gui.canvas.seconds2pos(transport.loopendtime), color(.6,.7,1,.7), color(
  }
  private void displayExactTimePoint(float t, float chroma)
  {
    float x = pos.x + (t-starttime)/(endtime-starttime) * size.x;
    line(x, pos.y+size.y/2, x,pos.y+size.y);
    text(""+roundBy(t,chroma),x,pos.y+size.y/2);
  }
  private void displayRoundedTimePoint(float t, float chroma)
  {
    t=roundBy(t,chroma);
    float x = pos.x + (t-starttime)/(endtime-starttime) * size.x;
    line(x, pos.y+size.y/2, x,pos.y+size.y);
    text(""+t,x,pos.y+size.y/2);
  }
};

float roundBy(float x, float chroma)
{
  return round(x/chroma)/(1/chroma);
}
```

transport.pde

```
public class Transport{
  float starttime, curtime;
  float loopstarttime, loopendtime;
  boolean playing,looping;
  ArrayList tempmidiinput;
  Transport()
  {
    tempmidiinput = new ArrayList();
```

```
  }
  void run()
  {
    if(playing && curtime>session.endTime())
      stop();
    if(playing && looping && curtime>loopendtime)
      stop();
  }
  void remotePlay()
  {
    playing=true;
    session.save();
  }
  void remoteStop()
  {
    println("remoteStop");
    playing=false;
    gui.inspector.recordbutton.active=false;
    if(tempmidiinput.size()>0)
    {
      float firsttime=((Coord)tempmidiinput.get(0)).x;
      float lasttime=((Coord)tempmidiinput.get(tempmidiinput.size()-1)).x;
      EnvRegion envregion = (EnvRegion)gui.inspector.region;
      envregion.removeBreakpointsInTimeRange(firsttime,lasttime);
      for(int imidibreakpoint=0; imidibreakpoint<tempmidiinput.size(); imidibreakpoint++)
      {
        Coord tempbreakpoint = (Coord) tempmidiinput.get(imidibreakpoint);
        envregion.addBreakpoint(new Breakpoint(envregion, makeFloatInputFieldTargetArray(tempbreakpoint.x, tempbreakpoint.y)));
      }
      tempmidiinput = new ArrayList();
      envregion.sortBreakpointsByTime();
      ((EnvRegionView)envregion.view).updateBreakpointsMinimumAndMaximum();
    }
  }
  void remoteLocate(float time)
  {
    println("remoteLocate: "+time);
    playing=false;
    curtime=time;
  }
  void remoteUpdatePlayPos(float time)
  {
    playing=true;
    curtime=time;
  }
  void play()
  {
    if(gui.inspector.recordbutton.active)
    {
      if(gui.inspector.region.isEnvRegion)
      {
        rohr.send("/record_midi", gui.inspector.region.id);
        return;
      }
      else
      {
        rohr.send("/record_audio", gui.inspector.region.id);
        return;
      }
    }
    rohr.send("/transport_play");
  }
  void stop()
  {
    rohr.send("/transport_stop");
  }
  void locate(float time)
  {
    rohr.send("/transport_locate", time);
  }
  void togglePlay()//spacebar
  {
    println("tog "+playing);
    if(!playing)
      play();
    else
      stop();
  }
};
```

xml.pde

```
import proxml.*;
XMLInOut xmlInOut;

Synth parseSynth(XMLElement element)
{
  debug(element.getName());
  Synth synth = new Synth(element.getAttribute("name"));
```

```
  for(int iparam = 0; iparam < element.countChildren(); iparam++)
  {
    XMLElement xmlparameter=element.getChild(iparam);
    debug(xmlparameter.getName());
    String name = xmlparameter.getAttribute("name");
    debug(name);
    switch(xmlparameter.getName().charAt(0))
    {
    case 's':
      synth.addParam(new StringParam(name, xmlparameter.getAttribute("value")));
      break;
    case 'a':
      debug("audioparam__value___"+xmlparameter.getAttribute("value"));
      AudioParam p=new AudioParam(name, xmlparameter.getFloatAttribute("value"), xmlparameter.getIntAttribute("isrouting")>0);
      synth.addParam(p);
      break;
    case 'n':
      synth.addParam(new NumberParam(name, xmlparameter.getFloatAttribute("value")));
      break;
    default:
      fail("unknown xml param type "+xmlparameter.getAttribute("name")+". s (string), p (filepath), a (audio), n (number) are allowed");
    }
  }
  return synth;
}

OP parseOP(XMLElement element, Region parent)
{
  debug("parse OP");
  String opname=element.getAttribute("name");
  int opnum=getOPNumByName(opname);
  if(opnum<0)
    fail("unknown op name "+opname);
  float arg=element.getFloatAttribute("arg");
  int channel=element.getIntAttribute("channel");
  boolean applyonregions=element.getAttribute("applytoregions").equals("true");
  boolean applyonbreakpoints=element.getAttribute("applytobreakpoints").equals("true");
  debug("parse OP__DONE");
  return new OP(opnum, arg, channel, applyonregions, applyonbreakpoints);
}
Breakpoint parseBreakpoint(XMLElement element, Region parent)
{
  debug(element.getName());
  float starttime = element.getFloatAttribute("starttime");
  float y1 = element.getFloatAttribute("y1");
  Breakpoint breakpoint = new Breakpoint(parent, makeFloatInputFieldTargetArray(starttime, y1));
  return breakpoint;
}
GroupRegion parseGroupRegion(XMLElement element, RegionWithChildren _parent)
{
  debug(element.getName());
  String name=element.getAttribute("name");
  int id= element.getIntAttribute("id");
  float starttime = element.getFloatAttribute("starttime");
  float duration  = element.getFloatAttribute("duration");
  GroupRegion region = new GroupRegion(_parent, name, id, makeFloatInputFieldTargetArray(starttime, duration));
  region.init();
  for(int i = 0; i < element.countChildren();i++)
  {
    XMLElement child = element.getChild(i);
    if(child.getName().equals("groupregion"))
    {
      region.addChild(parseGroupRegion(child, region));
    }
    else if(child.getName().equals("synthregion"))
    {
      region.addChild(parseSynthRegion(child, region));
    }
    else if(child.getName().equals("opregion"))
    {
      region.addChild(parseOPRegion(child, region));
    }
    else if(child.getName().equals("envregion"))
    {
      region.addChild(parseEnvRegion(child, region));
    }
    else if(child.getName().equals("audiosend"))
    {
      parseAudioSend(child, region);
    }
    else if(child.getName().equals("fade"))
    {
      parseFade(child, region);
    }
    else
      fail("illegal xml element "+child.getName()+" as child of "+element.getName());
  }
  return region;
}

SynthRegion parseSynthRegion(XMLElement element, RegionWithChildren _parent)
{
  debug(element.getName());
  String name=element.getAttribute("name");
  int id= element.getIntAttribute("id");
  float starttime = element.getFloatAttribute("starttime");
  float duration  = element.getFloatAttribute("duration");
  SynthRegion region = new SynthRegion(_parent, name, id, makeFloatInputFieldTargetArray(starttime, duration));
  region.init();
```

```
  for(int i = 0; i < element.countChildren();i++)
  {
    XMLElement child = element.getChild(i);
    if(child.getName().equals("groupregion"))
    {
      region.addChild(parseGroupRegion(child, region));
    }
    else if(child.getName().equals("synthregion"))
    {
      region.addChild(parseSynthRegion(child, region));
    }
    else if(child.getName().equals("opregion"))
    {
      region.addChild(parseOPRegion(child, region));
    }
    else if(child.getName().equals("envregion"))
    {
      region.addChild(parseEnvRegion(child, region));
    }
    else if(child.getName().equals("audiosend"))
    {
      parseAudioSend(child, region);
    }
    else if(child.getName().equals("synth"))
    {
      if(region.getSynth()==null)
      {
        Synth s=parseSynth(child);
        region.setSynth(s);
      }
      else
        fail("SynthRegion can not have more than 1 synth");
    }
    else if(child.getName().equals("fade"))
    {
      parseFade(child, region);
    }
    else
      fail("illegal xml element "+child.getName()+" as child of "+element.getName());
  }
  if(region.getSynth()==null)
    fail("SynthRegion must have 1 synth");
  return region;
}

OPRegion parseOPRegion(XMLElement element, RegionWithChildren _parent)
{
  debug(element.getName());
  String name=element.getAttribute("name");
  int id= element.getIntAttribute("id");
  float starttime = element.getFloatAttribute("starttime");
  float duration  = element.getFloatAttribute("duration");
  OPRegion region = new OPRegion(_parent, name, id, makeFloatInputFieldTargetArray(starttime, duration));
  region.init();
  for(int i = 0; i < element.countChildren();i++)
  {
    XMLElement child = element.getChild(i);
    if(child.getName().equals("groupregion"))
    {
      region.addChild(parseGroupRegion(child, region));
    }
    else if(child.getName().equals("synthregion"))
    {
      region.addChild(parseSynthRegion(child, region));
    }
    else if(child.getName().equals("opregion"))
    {
      region.addChild(parseOPRegion(child, region));
    }
    else if(child.getName().equals("envregion"))
    {
      region.addChild(parseEnvRegion(child, region));
    }
    else if(child.getName().equals("audiosend"))
    {
      parseAudioSend(child, region);
    }
    else if(child.getName().equals("op"))
    {
      if(region.getOP()==null)
        region.setOP(parseOP(child, region));
      else
        fail("OPRegion can not have more than 1 OP");
    }
    else if(child.getName().equals("fade"))
    {
      parseFade(child, region);
    }
    else
      fail("parseOPRegion: illegal xml element "+child.getName()+" as child of "+element.getName());
    if(region.getOP()==null)
      fail("OPRegion parsed without OP");
  }
  return region;
}
EnvRegion parseEnvRegion(XMLElement element, RegionWithChildren _parent)
{
  debug(element.getName());
  String name=element.getAttribute("name");
  int id= element.getIntAttribute("id");
```

```
  float starttime = element.getFloatAttribute("starttime");
  float duration  = element.getFloatAttribute("duration");
  float interpolation  = element.getFloatAttribute("interpolation");
  EnvRegion region = new EnvRegion(_parent, name, id, makeFloatInputFieldTargetArray(starttime, duration), interpolation);
  region.init();
  for(int i = 0; i < element.countChildren();i++)
  {
    XMLElement child = element.getChild(i);
    if(child.getName().equals("breakpoint"))
    {
      region.addBreakpoint(parseBreakpoint(child, region));
    }
    else
      fail("illegal xml element "+child.getName()+" as child of "+element.getName());
  }
  return region;
}

void parseAudioSend(XMLElement element, RegionWithChildren region)
{
  debug(element.getName());
  int destinationID = element.getIntAttribute("destinationID");
  float level = element.getFloatAttribute("level");
  region.addAudioSend(new AudioSend(destinationID, level));
}

void parseFade(XMLElement element, RegionWithChildren region)
{
  debug(element.getName());
  float intime = element.getFloatAttribute("intime");
  float incurve = element.getFloatAttribute("incurve");
  float outtime = element.getFloatAttribute("outtime");
  float outcurve = element.getFloatAttribute("outcurve");
  region.setFade(new Fade(intime, incurve,outtime,outcurve));
}
void xmlEvent(XMLElement element){ // is called by proXML loadElement - loads a session xml element
  debug(element.getName());
  sessionmanager.parseSession(element);
}
```

Anhang E

Quellcode SuperCollider

Dies ist der SuperCollider-Quellcode zum Zeitpunkt der Realisation von „Studioluft“. Die jeweils aktuelle Version ist bei mir auf Anfrage per Email erhältlich.

Y.sc

```
Y{classvar address_to_processing, <>sessionidentifier, <session, <buffers;
*init {
var commands;
this.server.waitForBoot({
YGraph.init;
YImpex.init;
YSynthDatabase.init;
YSynthDatabase.print;
YAnalysisDatabase.init;
YAnalysisDatabase.print;
this.connect_session_to_processing;
"Y init done".postln;
},225);
buffers=[];
YTransport.init;
address_to_processing = NetAddr("127.0.0.1", 17000);
commands = [
["/connect_session", {arg t,r,m; this.connect_session_to_processing}],
["/transmit_synthdatabase", {arg t,r,m; YSynthDatabase.transmit}],
["/transport_play", {arg t,r,m; YTransport.remotePlay(m[1],m[2])}],
["/transport_stop", {YTransport.remoteStop}],
["/transport_locate", {arg t,r,m; YTransport.remoteLocate(m[1])}],
["/record_audio", {arg t,r,m; YImpex.remoteRecordAudio(m[1])}],
["/record_midi",  {arg t,r,m; YImpex.remoteRecordMidi(m[1])}],
["/export", {arg t,r,m; YImpex.remoteExport(m[1])}],
["/analyse", {arg t,r,m; YImpex.remoteAnalyse(m[1],m[2],m[3],m[4],m[5])}],
["/sampleeditor", {arg t,r,m; YImpex.remoteSampleeditor(m[1])}]
];
commands.do({arg c; OSCresponder(nil, c[0], c[1]).add});
MIDIIn.connect(0, 3);
CCResponder({ |src, chan, num, val| YImpex.midiCallback(val)});
}
*server{^Server.default}
*debug{^true}
*send{arg command, a1, a2, a3, a4;
if(command.asString=="/transport_updateplaypos", {}, {(">>>sending OSC:"+command).postln});
if(a1==nil, {^address_to_processing.sendMsg(command)});
if(a2==nil, {^address_to_processing.sendMsg(command, this.convertOSCArgument(a1))});
if(a3==nil, {^address_to_processing.sendMsg(command, this.convertOSCArgument(a1), this.convertOSCArgument(a2))});
if(a4==nil, {^address_to_processing.sendMsg(command, this.convertOSCArgument(a1), this.convertOSCArgument(a2), this.convertOSCArgument(a3))});
^YError("Y can't send message with more than 3 args:"+command);
```

```
}
*convertOSCArgument{arg argument;
if(argument.isNumber, {^argument.asFloat});
if(argument.isString, {^argument});
if(argument.class==Symbol, {^argument.asString});
if(argument==true, {^1.0});
if(argument==false, {^0.0});
^YError.new("Y can't send unsupported type arg "+argument+"of class"+argument.class);
}
*connect_session_to_processing{
var petze, applet;
"Y got message /connect_session".postln;
petze=YJavaObject("SwingOSCPetze");
applet=YJavaObject.newFrom(petze, "getApplet");
session = YJavaObject.newFrom(applet,"getSession");
{this.checkBuffers}.fork;
this.send("/connect_session");
petze.destroy;
applet.destroy;
}
*checkBuffers{
var path, bufnum, javabuffer;
"checkBuffers...".postln;
session.countBuffers_.do({arg ibuffer;
javabuffer=YJavaObject.newFrom(session,\getBuffer,ibuffer);
bufnum=javabuffer.getBufnum_;
path=javabuffer.getPath_;
("Buffer"+bufnum+"___"+path).postln;
if(this.bufferByBufnum(bufnum).isNil, {
buffers=buffers.add(Buffer.read(Y.server,path,0,-1,nil,bufnum));
},{
if(this.bufferByBufnum(bufnum).path!=path, {
buffers=buffers.add(Buffer.read(Y.server,path,0,-1,nil,bufnum));
})
})
});
"checkBuffers done".postln;
}
*bufferByBufnum{arg bufnum;
buffers.do({arg buffer;
if(buffer.bufnum==bufnum,{^buffer});
});
^nil
}
*getRegion{arg id_or_name;
var region;
if(id_or_name.isNumber,{
region=YJavaObject.newFrom(this.session,\getRegionByID,id_or_name.floor);
},{
region=YJavaObject.newFrom(this.session,\getRegionByName,id_or_name.asString);
});
^region
}
*getRoot{
^YJavaObject.newFrom(this.session,\getRootRegion);
}
}
```

YAnalysis.sc

```
YAnalysisDatabase{
classvar <entries;
*init{
entries=[YAnalysis_Peak, YAnalysis_Attacks];
entries.do({arg entry; entry.init});
}
*entryByName{arg name;
entries.do({arg entry;
if(entry.name.asString==name.asString, {^entry})
});
^nil
}
*print{
"\n.....................................................".postln;
"---------     YAnalysisDatabase    -----------------\n\\...................................../".postln;
entries.do({arg entry; entry.print;})
}
}

YAnalysis{
classvar <javaregion, <starttime, <duration, <soundfile, <sampledata, <envregion;
*getDetails{arg id;
var soundFile;
("reading"+YImpex.exportedpath).postln;
soundfile = SoundFile.openRead(YImpex.exportedpath.asString);
javaregion = Y.getRegion(id);
starttime  = javaregion.heavyStartTime_;
duration   = javaregion.heavyDuration_;
```

```
sampledata = FloatArray.series(soundfile.numFrames,0,0);
soundfile.readData(sampledata);
}
*print{
("~~~~~~~~~~"+this.name+"~~~~~~~~~~~~~~~~~~~~~~").postln;
}
}
YAnalysis_Peak : YAnalysis{
classvar <name="peak";
*init{
}
*perform{arg id, interval;
var peak=0;
{
this.getDetails(id);
envregion = javaregion.newChildEnv("peak",1);
sampledata.do({arg y,i;
if(y.abs>peak, {peak=y.abs});
if(i%(interval*44100)<1,{
envregion.newBreakpoint(i/44100+starttime,peak);
peak=0;
});
});
}.fork
}
}
YAnalysis_Attacks : YAnalysis{
classvar <name="attacks";
*init{
}
*perform{arg id, smooth=0, slopethreshold=0.2, mintime=0.05;
var downsampled, prevy, ableitung, previ, maxy=0;
{
this.getDetails(id);
111.postln;
envregion = javaregion.newChildEnv("attacks",0);
downsampled=FloatArray.newClear(sampledata.size/10);
prevy=sampledata[0];
222.postln;
downsampled.size.do({arg i;//downsample and highpass
downsampled[i]=0.5*sampledata[i*10]-(0.5*prevy);
prevy=downsampled[i];
});
prevy = abs(downsampled[0]);
downsampled.do({arg y,i;//peakmeter with fast attack and slow decay
downsampled[i] = (prevy*smooth) + ((1-smooth) * if(abs(y)>prevy, {abs(y)},{prevy*0.99}));
prevy=downsampled[i];
});
333.postln;
downsampled.do({arg y,i;//one more lowpass
downsampled[i]= 0.5*y+(0.5*prevy);
prevy=downsampled[i];
});
ableitung=Array.newClear(downsampled.size);
prevy = downsampled[0];
downsampled.do({arg y,i;//get derivative and maximum slope
ableitung[i]= y-prevy;
if(ableitung[i]>maxy, {maxy=ableitung[i]});
prevy=downsampled[i];
});
444.postln;
prevy = downsampled[0];
ableitung.do({arg y,i;//find attacks
//if((previ.isNil || (i-previ>200)) &&  geht nicht, bei || wird links und rechts ausgefuehrt
if( if(previ.isNil,{true},{i-previ>(mintime*4410)}) &&
(prevy > (maxy*slopethreshold)) &&
(prevy > y),{
envregion.newBreakpoint(max(0,i-1)/4410 + starttime, y/maxy);
previ=i;
});
prevy=ableitung[i];
});
}.fork
}
}
```

YError.sc

```
YError{
*new{arg msg;
("YERROR!  "+msg).postln;
}
}
```

YGraph.sc

```
YGraph{
//the graph is built from scratch
//everytime the user hits play, export, etc..
//all YGraphRegion objects are destroyed
//immediately after transport stops.
classvar <regions, <delaycompensation;
*init{
delaycompensation=0.0;
this.clearRegions;
}
*clearRegions{
regions=[];
}
*regionByID{arg id;
regions.do({arg region;
if(id==region.id, {^region});
});
^nil
}
*build{arg functionToCallWhenReady;
{
"YGraph building regions...".postln;
this.buildHelper(Y.getRoot);
regions.postln;
"YGraph connecting audio sends...".postln;
this.connectAudioSends;
this.prepareSynths;
{
if(functionToCallWhenReady.isNil, {}, {functionToCallWhenReady.value});
}.defer(0.1);
}.fork
}
*play{arg playstarttime, functionToCallWhenReady;
"YGraph play...".postln;
regions.do({arg region; region.play(playstarttime)});
("listening to region "+regions[0].id).postln;
SystemClock.sched(delaycompensation, {
if(functionToCallWhenReady.isNil, {}, {functionToCallWhenReady.value});
nil});
}
*stop{
if(regions.size==0, {^false});
"YGraph stopping...".postln;
regions.do({arg region; region.stop});
this.clearRegions;
}
*buildHelper{arg javaregion;
this.buildAddRegion(javaregion);
if(javaregion.isEnv_<0.5, {
("nchildren:"+javaregion.countChildren_).postln;
javaregion.countChildren_.do({arg ichild;
this.buildHelper(JavaObject.newFrom(javaregion, \getChild, ichild));
});
});
}
*buildAddRegion{arg javaregion;
"buildAddRegion".postln;
if(javaregion.isGroup_>0, {regions=regions.add(YGraphGroupOrOPRegion.new(javaregion))});
if(javaregion.isSynth_>0, {regions=regions.add(YGraphSynthRegion.new(javaregion))});
if(javaregion.isOP_>0, {regions=regions.add(YGraphGroupOrOPRegion.new(javaregion))});
if(javaregion.isEnv_>0, {regions=regions.add(YGraphEnvRegion.new(javaregion))});
}
*connectAudioSends{
var destRegion;
regions.do({arg region;
region.class.postln;
if(region.class!=YGraphEnvRegion, {
region.audiosends.do({arg send;
("send----   "+region.id+"->"+send.destID).postln;
if(send.destID== -1, {
destRegion=this.regionByID(region.parentid)
},{
if(send.destID== -2 ,{
destRegion=regions[0]
},{
destRegion=this.regionByID(send.destID)
});
});
if(destRegion!=nil, {
destRegion.mixbusproxy.add(region.outproxy * send.level)
});
});
});
});
}
*prepareSynths{
"prepareSynths".postln;
regions.do({arg region;
if(region.class==YGraphSynthRegion, {
region.synth.getParamsFromJava(YJavaObject.newFrom(region.javaregion,\getSynth));
})
});
}
}
```

```
YGraphAudioSend{
var <destID, <level;
init{arg in_destID, in_level;
destID=in_destID;
level=in_level;
}
}

YGraphRegion{//ABSTRACT
var <id, <starttime, <duration, <outproxy, <parentid;
initHead{arg javaregion;
id = javaregion.getID_;
starttime = javaregion.lightStartTime_;
duration = javaregion.lightDuration_;
parentid = javaregion.getParentID_;
}
stop{
"YGraphRegion.stop".postln;
outproxy.clear;
}
endtime{
^starttime+duration
}
javaregion{
^JavaObject.newFrom(Y.getRoot, \getRegionByID, id)
}
}

YGraphRegionWithChildren : YGraphRegion{
var <mixbusproxy, <audiosends, <fadeintime, <fadeincurve, <fadeouttime, <fadeoutcurve;
initFade{arg javaregion;
fadeintime=javaregion.fadeInTime_;
fadeincurve=javaregion.fadeInCurve_.max(0.01);
fadeouttime=javaregion.fadeOutTime_;
fadeoutcurve=javaregion.fadeOutCurve_.max(0.01);
}
buildAudioSends{arg javaregion;
var nsends=javaregion.countAudioSends_;
"buildAudioSends".postln;
audiosends=[];
nsends.do({arg isend;
var send, destID, level;
send = JavaObject.newFrom(javaregion, \getAudioSend, isend);
destID = send.getAbsoluteID_;
level = send.getLevel_;
audiosends = audiosends.add(YGraphAudioSend.new.init(destID,level));
});
"buildAudioSends done".postln;
}
play{arg playstarttime;
var startdelay, stopdelay, fadephase;
("play region id"+id+" ____starttime:"+starttime+"___duration:"+duration).postln;
("___playstarttime:"+playstarttime).postln;
if(starttime+duration <= playstarttime, {^false});//region lies in past
startdelay = max(0,starttime-playstarttime);
stopdelay = starttime+duration - playstarttime;
if(playstarttime>starttime,{//region has already started
if(playstarttime<(starttime+fadeintime), {//is just fading in
//approximate rest of fade in curve
fadephase = (playstarttime-starttime)/fadeintime;
fadeintime = starttime+fadeintime-playstarttime;
outproxy[2] = \filter->{arg in; in * (Line.ar(fadephase**fadeincurve,1,fadeintime)**fadeincurve)}
},{
outproxy[2] = \filter->{arg in; in};
})
},{
//region starts now or in future
SystemClock.sched(startdelay,
{outproxy[2] = \filter->{arg in; in * (Line.ar(0,1,fadeintime)**fadeincurve)};
nil});
});
if(playstarttime>(starttime+duration-fadeouttime), {//is just fading out
//approximate rest of fade out curve
fadephase = (playstarttime-(starttime+duration-fadeouttime))/fadeouttime;
fadeouttime = starttime+duration-playstarttime;
outproxy[2] = \filter->{arg in; in *(Line.ar(fadephase**fadeoutcurve,1,fadeouttime)**fadeoutcurve*(-1)+1)}
},{//fade out starts now or in the future
SystemClock.sched(stopdelay-fadeouttime,
{outproxy[2] = \filter->{arg in; in * (Line.ar(0,1,fadeouttime)**fadeoutcurve*(-1)+1)};
nil});
});
}
stop{
"YGraphRegionWithChildren.stop".postln;
super.stop;
mixbusproxy.clear
}
}

YGraphGroupOrOPRegion : YGraphRegionWithChildren{
*new{arg javaregion;
^super.new.init(javaregion)
}
init{arg javaregion;
this.initHead(javaregion);
this.initFade(javaregion);
mixbusproxy = NodeProxy.audio(Y.server,1);
outproxy = NodeProxy.audio(Y.server,1);
outproxy.source = mixbusproxy; // oder geht auch outputproxy = mixbusproxy?
outproxy[2] = \filter->{arg in; 0};
this.buildAudioSends(javaregion);
```

```
^this
}
}

YGraphSynthRegion : YGraphRegionWithChildren{
var <synth;
*new{arg javaregion;
^super.new.init(javaregion);
}
init{arg javaregion, javasynth;
this.initHead(javaregion);
this.initFade(javaregion);
synth = YSynthDatabase.entryByName(javaregion.getSynthName_).new.init;
javasynth = JavaObject.newFrom(javaregion, \getSynth);
("nparams="+javasynth.countParams_).postln;
mixbusproxy = NodeProxy.audio(Y.server,1);
outproxy = NodeProxy.audio(Y.server,1);
outproxy.source = mixbusproxy;
outproxy[2] = \filter->{arg in; 0};
this.buildAudioSends(javaregion);
^this
}
play{arg playstarttime;
super.play(playstarttime);
synth.play(playstarttime, this)
}
}

YGraphEnvRegion : YGraphRegion{
var breakpoints, interpolation;
*new{arg javaregion;
^super.new.init(javaregion);
}
init{arg javaregion;
this.initHead(javaregion);
this.buildBreakpoints(javaregion);
interpolation=javaregion.getInterpolation_;
outproxy = NodeProxy.audio(Y.server,1);
^this
}
buildBreakpoints{arg javaregion;
var nbreakpoints, prevx;
"buildBreakpoints".postln;
nbreakpoints=javaregion.countBreakpoints_;
breakpoints=[];
nbreakpoints.do({arg ibreakpoint;
var javabreakpoint, x, y;
javabreakpoint = JavaObject.newFrom(javaregion, \getBreakpoint, ibreakpoint);
x = javabreakpoint.lightStartTime_;
y = javabreakpoint.lightValueAt_(1);
if(x!=prevx, {breakpoints = breakpoints.add([x,y])});//important: avoid simultaneous
prevx=x;
});
"buildBreakpoints done".postln;
}
play{arg playstarttime;
var time, timenext, y, ynext, phase, ibreakpoint=0;
("play region id"+id).postln;
if(breakpoints.size==0, {outproxy.source = 0; ^false});
if(breakpoints.size==1, {outproxy.source = breakpoints.first[1]; ^false});
if(breakpoints.last[0] <= playstarttime, {//breakpoints lie in past
outproxy.source = breakpoints.last[1];
^false
});
if(breakpoints.first[0] > playstarttime, {//breakpoints lie in future
outproxy.source = breakpoints.first[1];
timenext = breakpoints.first[0];
},{
//skip past
while({(ibreakpoint < (breakpoints.size-1)) && (breakpoints[ibreakpoint+1][0] <= playstarttime)},{
ibreakpoint=ibreakpoint+1;
});
//immediately play (last part of) current
if(ibreakpoint < (breakpoints.size-1), {
time = breakpoints[ibreakpoint][0];
y = breakpoints[ibreakpoint][1];
timenext = breakpoints[ibreakpoint+1][0];
ynext = breakpoints[ibreakpoint+1][1];
phase = (playstarttime-time)/(timenext-time);
y = y +(phase*(ynext-y));
outproxy.source = {Line.ar(0,1, timenext-playstarttime)**interpolation*(ynext-y)+y};
ibreakpoint = ibreakpoint+1;
});
});
//schedule future breakpoints from each one to the next
//because the scheduler size is limited, and it is accurate enough
SystemClock.sched(timenext-playstarttime, {
if(ibreakpoint >= (breakpoints.size-1), {nil},{
time = breakpoints[ibreakpoint][0];
y = breakpoints[ibreakpoint][1];
timenext = breakpoints[ibreakpoint+1][0];
ynext = breakpoints[ibreakpoint+1][1];
outproxy.source = {Line.ar(0,1, timenext-time)**interpolation*(ynext-y)+y};
ibreakpoint = ibreakpoint+1;
timenext-time;
});
});
}
}
```

YImpex.sc

```
YImpex{
classvar recbuf, proxy, midirecording, <exportedpath;//, <>id, <path, <starttime, <duration, exportproxy;
*init{
recbuf=Buffer.alloc(Y.server, 44100*100, 1, nil, 999);
proxy=NodeProxy.audio(Y.server,1);
midirecording=false;
}

*remoteRecordAudio{arg parentID;
this.recordAudio(parentID)
}
*remoteRecordMidi{arg parentID;
this.recordMidi(parentID)
}
*remoteExport{arg id;
this.export(id)
}
*remoteAnalyse{arg methodname, id, arg1, arg2, arg3;
this.analyse(methodname,id, arg1, arg2, arg3)
}
*remoteSampleeditor{arg id;
//TODO
}
//---------------------- end remote
*writeRecBufToFile{arg duration, prefix;
var path, bufid;
{
path=Y.session.getTempFilePath_(prefix);
("writing "+path).postln;
recbuf.write(path, \RIFF, \int16, duration*44100, 0, false, {{exportedpath=path}.defer(2);nil})
}.fork
}
*importAsBufferAndSynthRegion{arg path, parentID, starttime, duration, synthregionname;
var bufid, javaregion, javachild, javasynth, javaparam;
{
bufid=Y.session.smallestFreeBufnum_(0);
("adding"+path+"as buffer number"+bufid).postln;
("new synth region named '"++synthregionname++"' as child of"+parentID+"at starttime:"+starttime+"duration:"+duration).postln;
javaregion = Y.getRegion(parentID);
javaregion.bla;
javachild=javaregion.newChildSynth(synthregionname, starttime, duration, "PlayBuf");
javachild.bla;
javasynth=YJavaObject.newFrom(javachild,\getSynth);
javaparam=YJavaObject.newFrom(javasynth,\getParam ,0);
javaparam.setFloat(bufid);
Y.send("/new_buffer",path);
Y.checkBuffers;
Y.session.saveForced;
}.fork
}
*recordAudio{arg parentID;
var starttime, duration;
{
"recording....".postln;
starttime  = YTransport.playstarttime;
"build graph now....".postln;
YGraph.build({
proxy.source={RecordBuf.ar(AudioIn.ar(1),999)};
YGraph.regions[0].outproxy.play;
YTransport.actuallyStartPlaying;
Routine{while({YTransport.playing},{0.0123.yield});
"recording stopped".postln;
proxy.free;
duration = YTransport.timewhenstopped - starttime;
this.writeRecBufToFile(duration, "rec");
Routine{while({exportedpath.isNil},{0.0123.yield});
this.importAsBufferAndSynthRegion(exportedpath, parentID, starttime, duration, "rec")
}.play
}.play;
});
}.fork
}
*export{arg id;
var javaregion, starttime, duration;
exportedpath=nil;
{
"exporting....".postln;
javaregion = Y.getRegion(id);
starttime  = javaregion.heavyStartTime_;
duration   = javaregion.heavyDuration_;
YTransport.locate(starttime);
"build graph now....".postln;
YGraph.build({
proxy.source = {RecordBuf.ar(YGraph.regionByID(id).outproxy.ar,999)};
YGraph.regionByID(id).outproxy.play;
YTransport.actuallyStartPlaying;
SystemClock.sched(duration,{
proxy.free;
YTransport.stop;
"exporting done".postln;
this.writeRecBufToFile(duration, "exp", nil);
nil
})
```

```
})
}.fork
}
*analyse{arg methodname, id, arg1, arg2, arg3;
("analysing region"+id+"with method"+methodname+arg1+arg2+arg3).postln;
this.export(id);
Routine{while({exportedpath.isNil},{0.0123.yield});
if(YAnalysisDatabase.entryByName(methodname).isNil,{^YError.new("unknown analysis name"+methodname)});
YAnalysisDatabase.entryByName(methodname).perform(id, arg1, arg2, arg3);
}.play
}
*recordMidi{arg parentID;
{
"recording midi....".postln;
YGraph.build({
YTransport.actuallyStartPlaying;
midirecording=true;
Routine{while({YTransport.playing},{0.0123.yield});
"recording midi stopped".postln;
midirecording=false;
}.play
});
}.fork
}
*midiCallback{arg val;
if(midirecording && YTransport.playing, {val.postln; Y.send("/recorded_midi_breakpoint", YTransport.curtime, val/127)});
}
}
```

YJavaObject.sc

```
YJavaObject : JavaObject{
doesNotUnderstand { arg selector ... args;
var selStr;
selStr = selector.asString;
if( selStr.find("new") == 0, {
^YJavaObject.newFrom(this,selector, *args);
},{
^super.doesNotUnderstand(selector, *args)
});
}
}
```

YSynthDatabase.sc

```
YParam{var <name, <>value;
init{arg in_name, in_value;
name=in_name;
value=in_value;
}
set{arg newvalue;
value=newvalue;
}
print{
(name+"             type:"++this.class+"             value:"++value).postln;
}
}

YNumberParam : YParam{
*new{arg in_name, in_value;
^super.new.init(in_name, in_value);
}
print{
("NumberParam  "+name+"   "+value).postln
}
}
YAudioParam : YParam{
*new{arg in_name, in_value;
^super.new.init(in_name, in_value);
}
print{
("AudioParam  "+name+"   "+value).postln
}
}
```

```
YSynthDatabase{classvar <entries;
*init{
entries=[YSynth_WhiteNoise, YSynth_BPF, YSynth_PlayBuf];
entries.do({arg entry; entry.initClass});
}
*entryByName{arg synthname;
entries.do({arg entry;
if(entry.synthname.asString==synthname.asString, {^entry})
});
^nil
}
*print{
"\n_____________________________________________________\n||||||||     YSynthDatabase    ||||||||||||||||||||||||||||||\n\\_____________________________
entries.do({arg entry; entry.print;})
}
*transmit{
var petze, applet, javadatabase;
"transmitting SynthDatabase".postln;
petze=JavaObject("SwingOSCPetze");
applet=JavaObject.newFrom(petze, "getApplet");
javadatabase = JavaObject.newFrom(applet, "getSynthDatabase");
entries.do({arg entry;
var javasynth;
javasynth=JavaObject.newFrom(applet, "newSynth", entry.synthname);
javadatabase.addSynth(javasynth);
entry.defaultparams.do({arg p;
if(p.class==YAudioParam, {javasynth.addParam(JavaObject.newFrom(applet, "newAudioParam", p.name, p.value, false))});
if(p.class==YNumberParam, {javasynth.addParam(JavaObject.newFrom(applet, "newNumberParam", p.name, p.value))});
});
javasynth.destroy;
});
javadatabase.remoteTransmitDone;
javadatabase.destroy;
petze.destroy;
applet.destroy;
"done".postln;
}
}

YSynthClass{ // ABSTRACT
var <params, <>outproxysource;
*new{
^super.new.init;
}
*load{
}
init{
params = this.class.defaultparams.copy;
params.do({arg p,i; params[i]=p.copy});
"YSynthClass instance init".postln
}
play{
^YError.new("this is the play method of the abstract synth class. if you read this, you forgot to implement the method play in your synth")
}
getParamsFromJava{arg javasynth;
var javaparam;
("synth"+javasynth.getName_+"has"+params.size+"params:"+params).postln;
params.do({arg param,iparam;
javaparam=JavaObject.newFrom(javasynth,\getParam,iparam);
("param"+iparam+"type="+javaparam.getType_.asAscii).postln;
if(javaparam.getType_==$n.ascii,{
param.value = javaparam.getFloat_;
});
if(javaparam.getType_==$a.ascii,{
if(javaparam.getIsRouting_>0,{
param.value = YGraph.regionByID(javaparam.getValue_.floor).outproxy
},{
param.value = javaparam.getValue_
});
});
});
}
*print{
(">>>>>>>>"+this.synthname+"<<<<<<<<<<<<<<<<<<<<<").postln;
this.defaultparams.do({arg p; p.print})
}
}

YSynth_WhiteNoise : YSynthClass{
classvar <>synthname, <>defaultparams;
*initClass{
this.synthname = "WhiteNoise";
this.defaultparams = [YAudioParam.new("amp", 0.2)];
}
init{
super.init;
"WhiteNoise init".postln
}
play{arg playstarttime, region;
region.outproxy[1]=\filter->{arg in; WhiteNoise.ar(params[0].value)}
}
}

YSynth_BPF : YSynthClass{
classvar <>synthname, <>defaultparams;
*initClass{
this.synthname = "BPF";
this.defaultparams = [YAudioParam.new("freq", 1000), YAudioParam.new("q", 20)];
```

```
}
init{
super.init;
}
play{arg playstarttime, region;
region.outproxy[1]=\filter->{arg in; BPF.ar(in, params[0].value.max(1), (1.0/params[1].value.max(3)).min(20000/params[0].value.max(1)))}
}
}
YSynth_PlayBuf : YSynthClass{
classvar <>synthname, <>defaultparams;
*initClass{
this.synthname = "PlayBuf";
this.defaultparams = [YNumberParam.new("bufnum", 0), YAudioParam.new("offset", 0), YAudioParam.new("rate", 1)];
}
init{
super.init;
}
play{arg playstarttime, region;
var startdelay, stopdelay, trickoffset=0;
if(region.endtime <= playstarttime, {^false});//region lies in past
startdelay = max(0,region.starttime-playstarttime);
stopdelay = region.endtime - playstarttime;
("playbuf offset value="+params[1].value).postln;
("playbuf rate value="+params[2].value).postln;
if(params[2].value.isNumber && (region.starttime<playstarttime),{
trickoffset = (playstarttime-region.starttime)*params[2].value;
});
SystemClock.sched(startdelay,{region.outproxy[1]=\filter->{arg in; PlayBuf.ar(1, params[0].value, params[2].value, -1, (params[1].value+trickoffset)*44
}
}
```

YTransport.sc

```
YTransport{classvar  <>curtime, <>playstarttime, <timewhenstopped, <>playing, routine;
*init {
playing=false;
curtime=0;
playstarttime=0;
routine=Routine({
while({playing},{
curtime=curtime+0.005;
Y.send("/transport_updateplaypos",curtime);
0.005.yield;
});
curtime=playstarttime;
Y.send("/transport_locate",curtime);
Y.send("/transport_setplaying",0);
});
}
*remotePlay{
"remotePlay".postln;
this.play;
}
*remoteStop{
"remoteStop".postln;
this.stop;
}
*remoteLocate{arg time;
("remoteLocate"+time).postln;
this.stop;
curtime=time;
playstarttime=time;
Y.send("/transport_locate",curtime);
}
*locate{arg time=0;
this.stop;
curtime=time;
playstarttime=time;
Y.send("/transport_locate",curtime);
}
*play{
if(curtime.isNil,{curtime=0});
playstarttime=curtime;
("playstarttime:"+playstarttime).postln;
routine.stop;
routine.reset;
SystemClock.clear;
YGraph.build({this.actuallyStartPlaying; YGraph.regions[0].outproxy.play;})
}
*actuallyStartPlaying{
playing=true;
YGraph.play(playstarttime ,{
Y.send("/transport_play");
SystemClock.sched(0.2,{routine.play(SystemClock);nil});
});
}
*stop{
```

```
routine.stop;
routine.reset;
SystemClock.clear;
timewhenstopped=curtime;
curtime=playstarttime;
playing=false;
Y.send("/transport_locate",playstarttime);
Y.send("/transport_stop");
YGraph.stop;
}
}
```