NO CLERGY
GENERATION OF NOTATED MUSIC BASED ON AUDIENCE FEEDBACK IN REAL TIME USING PYTHON, MUSICXML AND GNU LILYPOND

by

Kevin Craig Baird
[defense date]
Major Professor: Cort Lippe

A dissertation submitted to the
Faculty of the Graduate School of
The State University of New York at Buffalo
in partial fulfillment of the requirements for the
degree of

Doctor of Philosophy

Department of Music

Dedication/Acknowledgements

Dedicated to Jennifer L. Cornish, who has the amazing graciousness necessary to put up with me.

This would not have been possible without the help of many people. Thanks to my advisor Cort Lippe, Director of the Lejaren Hiller Computer Music Studios at the University at Buffalo and to the rest of my committee: James Currie, Martha Hyde, and Jeff Stadelman. Thanks are also due to the late John Clough, who was the Slee Professor of Music Theory at the University at Buffalo.

Thanks to Erik Oña, currently at the University of Birmingham, England, especially for suggestions at the early stages of my earlier piece *No Cathedral*.

Thanks to Susan Fancher and Jason Crane for performer feedback, again largely relating to *No Cathedral*.

Thanks also to Jeff Higginbotham, Director of the Communication and Assistive Devices Laboratory, for the job, the network access, and the King Crimson.

Table of Contents

Abstract

The impetus behind *No Clergy* was the desire to afford audiences the ability to have an influence on a musical performance as it occurs. By itself, this is nothing new, but there are a few features of *No Clergy* that distinguish it from other pieces.

From the work of the late Earle Brown, especially *Available Forms I* and *Available Forms II*, it uses the idea of a random access ordering of material played by acoustic performers. Unlike Stockhausen's *Klavierstücke*, it gives the decision making power to someone other than the performer him or herself.

From the compositional environment at the University at Buffalo and the work of people like my advisor Cort Lippe, it takes the idea of modifying performance characteristics in real time and using more finely graded units than in Brown's pieces.

From an earlier project of mine called *No Cathedral*, it takes the idea of simultaneous acoustic performances of notation presented on computer screens. In *No Cathedral*, computer signals chose between images of entire pages of notation, whereas in *No Clergy*, each note is generated individually. It also provided the basis for the title, which incidentally has nothing in particular to do with religion.

From countless installations, it takes the idea of an immersive performance environment, in which audience members feel free to move around and feel like participants.

Part I. Setup and Performance

Table of Contents

Chapter 1. Setup

No Clergy runs on a Debian GNU/Linux [http://www.debian.org/] system on Intel x86 architecture (although the architecture shouldn't matter), using *Sarge*'s packages of

- apache (1.3.31-4)
- lilypond (2.2.5-1.pk) [1]
- python (2.3.4-5)
- python-xml (0.8.3-5)

Specific version numbers following in parentheses. This system (assumed to have the hostname `nibbler.med.buffalo.edu`) should be set up to serve web documents.

The performers (clarinet, soprano saxophone, and violin in the default configuration) should be in the center of an informal performance space, with one web browser equipped computer for each performer. The computers should be arranged such that the screens face outward and the performers face inward. There should be ample room for audience members to observe each performer's screen. Each performer should browse their web browser to `http://nibbler.med.buffalo.edu/noclergy/clar/` (in the base of the clarinettist), `http://nibbler.med.buffalo.edu/noclergy/sax/` (in the case of the saxophonist), or `http://nibbler.med.buffalo.edu/noclergy/vn/` (in the case of the violinist). Differing instrumentation will require creation of appropriate web directories and minor re-writing of some of the Python and bash scripts.

For the audience, one or more web browsers should be open to the `noclergy/` directory on the host (assumed to be `nibbler.med.buffalo.edu`).

The Python script `noclergy_score.py` contains a variable `localDTDpathS`, which is the pathname pointing to the local copy of the MusicXML DTD. This will need to be altered if the piece is run on a server other than mine.

[1] Available by adding

```
deb http://www.pedrokroeger.net/lilypond/ ./
```

to `/etc/apt/sources.list` and performing **`apt-get update; apt-get install lilypond`**

Chapter 2. Performance

When `~/setup.sh` is executed, initial random values are used to generate a first pass of musical notation for each of the performers. The semantic musical data are stored externally in a MusicXML [http://musicxml.org/] file, and are also rendered into an image file using GNU Lilypond [http://lilypond.org/].

During the performance, audience members input data into their web forms as desired. This data (along with internal data resulting from analysis of the results of the initial run of the script) is then used to create subsequent generations of musical data. Each performer plays the notation presented on his or her screen until the end of the piece.

The 2nd and later sets of musical notation are generated with the script `~/noclergy.sh`, which reads both the audience feedback data and the previous musical material stored in the MusicXML files in order to generate each successive page of notation for each performer. It also moves old files into a storage location, ensuring that any later runs of the script will be based on the freshest available data.

Ending the Piece. The premiere will simply run each script a fixed number of times, ending the piece when there are no more new pages of notation. Other performances could simply loop, or have a cutoff condition (ending the piece when the rest-to-note ratio rises above a certain point, for example). The looping option would be especially appropriate for alterations of the piece to use machine-generated sounds (Csound, or Max/MSP, for example), rather than live acoustic performances.

Part II. Sample Notated Output

Includes inline print-quality images of output for each instrument, however many pages are required for a full run of the piece. I have ten pages for each instrument to demonstrate multi-page pagination with something like the right page count.

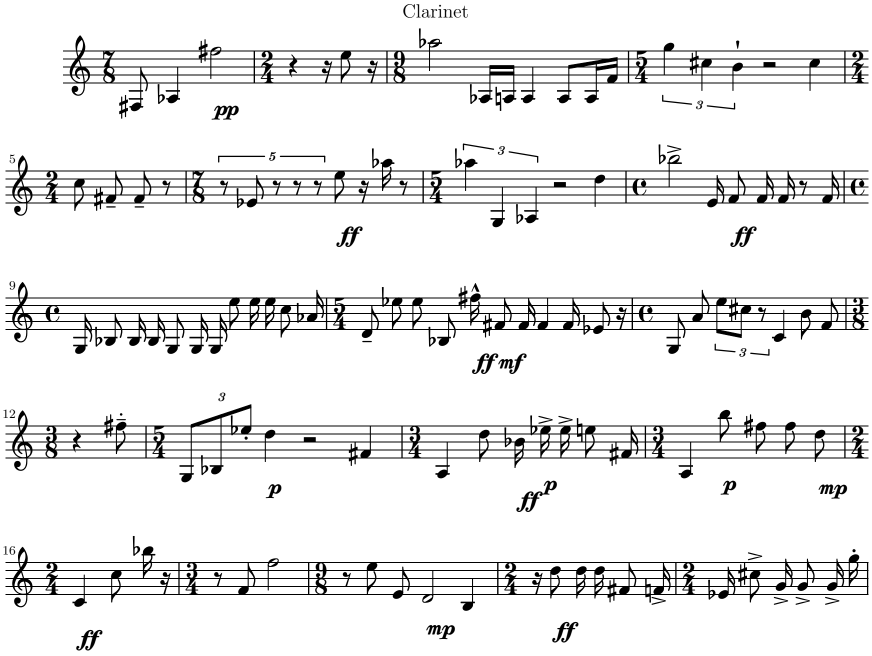
These samples use the values of `tupletpc = 5`, `restpc = 5`, `artpc = -3`, and `dynpc = -3` in the file `/var/www/noclergy/feedback.html` described later in its own section. Therefore, the parts should demonstrate a slow progression such that the later pages have much higher ratios of tuplets to non-tuplets and rests to notes, and a more moderate decrease in the number of articulatory and dynamic markings.

Table of Contents

Chapter 3. Clarinet

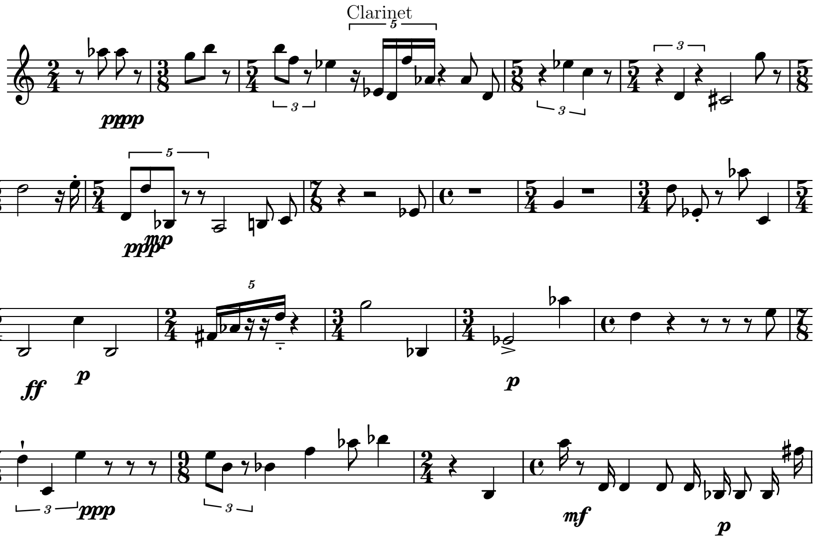
No Clergy

Kevin C. Baird



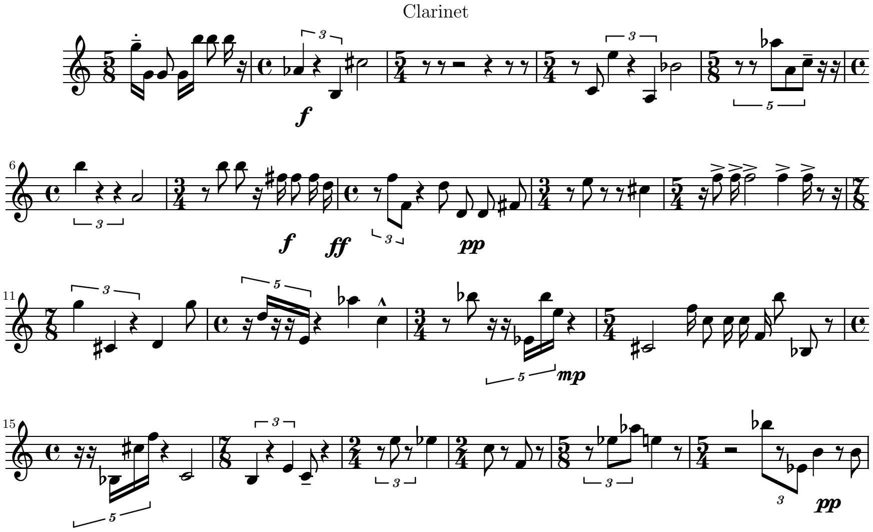
No Clergy

Kevin C. Baird



No Clergy

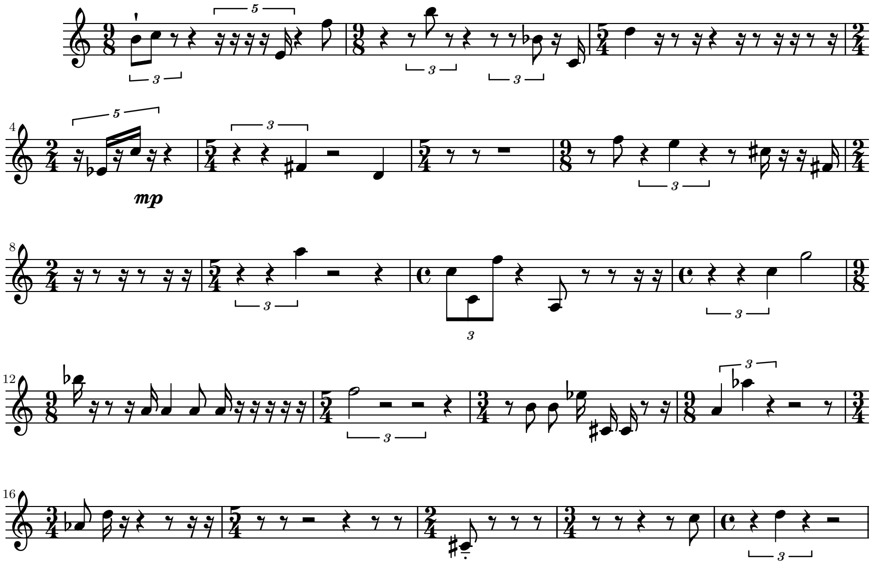
Kevin C. Baird



No Clergy

Kevin C. Baird

Clarinet

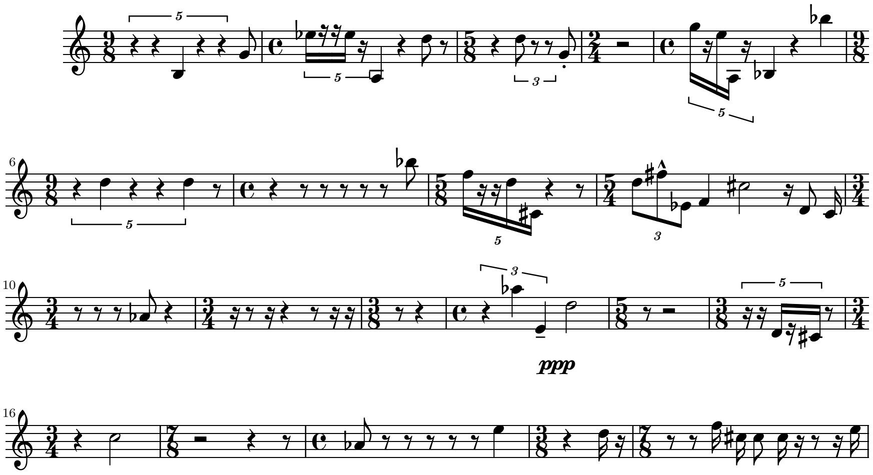


1

No Clergy

Kevin C. Baird

Clarinet

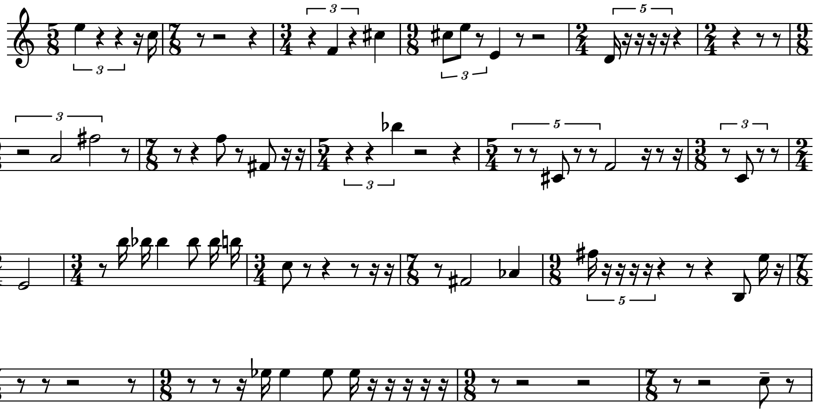


1

No Clergy

Kevin C. Baird

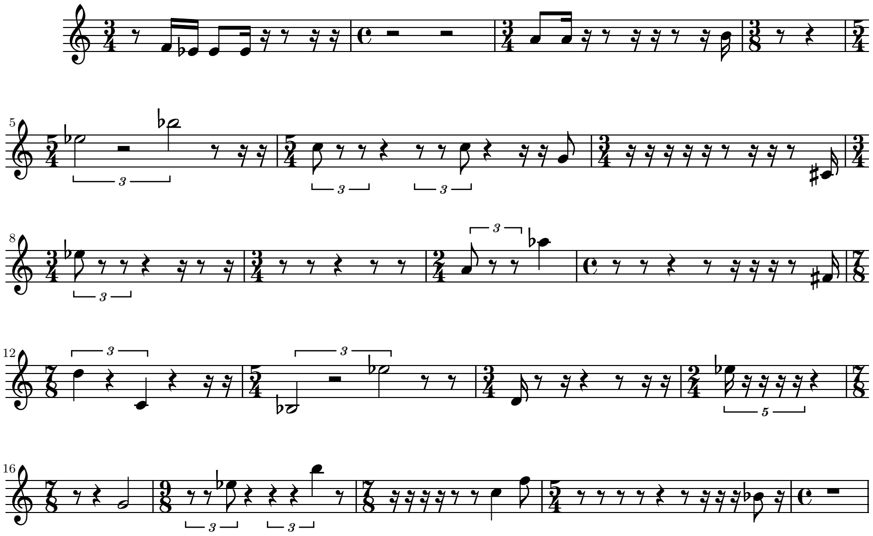
Clarinet



No Clergy

Kevin C. Baird

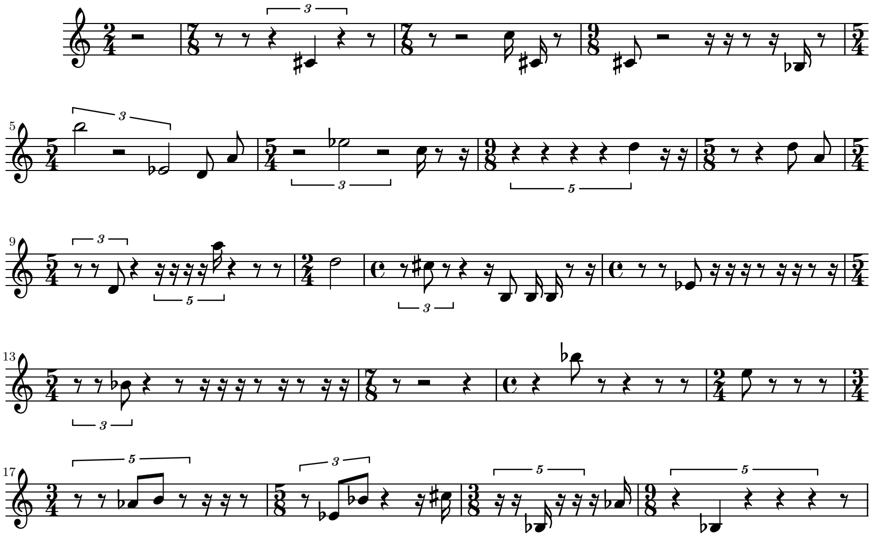
Clarinet



No Clergy

Kevin C. Baird

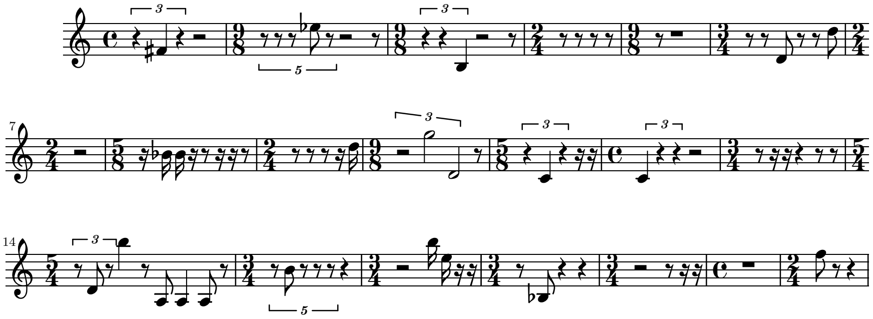
Clarinet



No Clergy

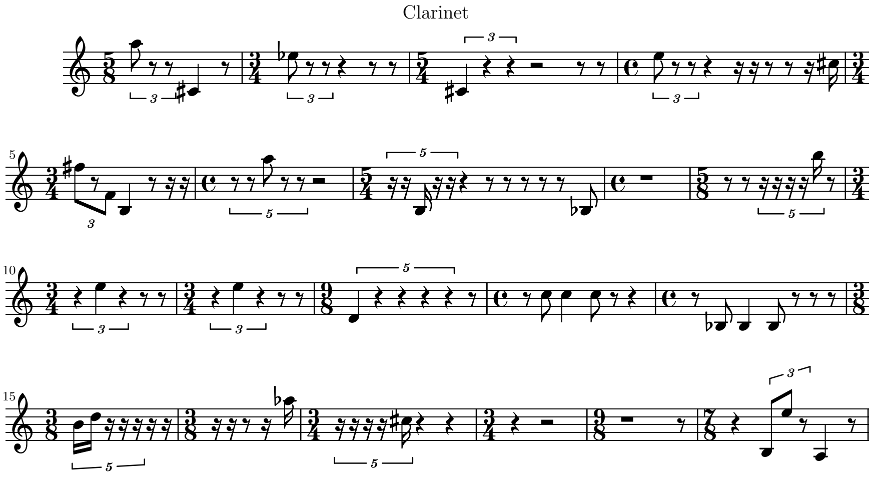
Kevin C. Baird

Clarinet



No Clergy

Kevin C. Baird

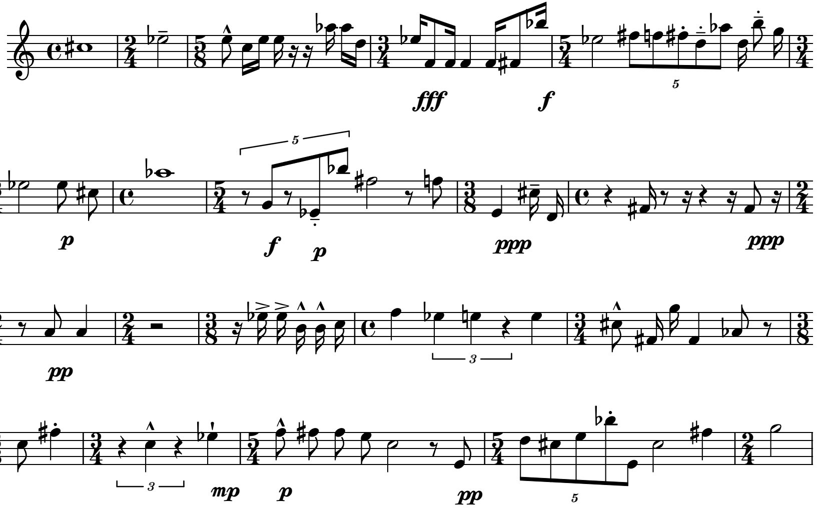


Chapter 4. Saxophone

No Clergy

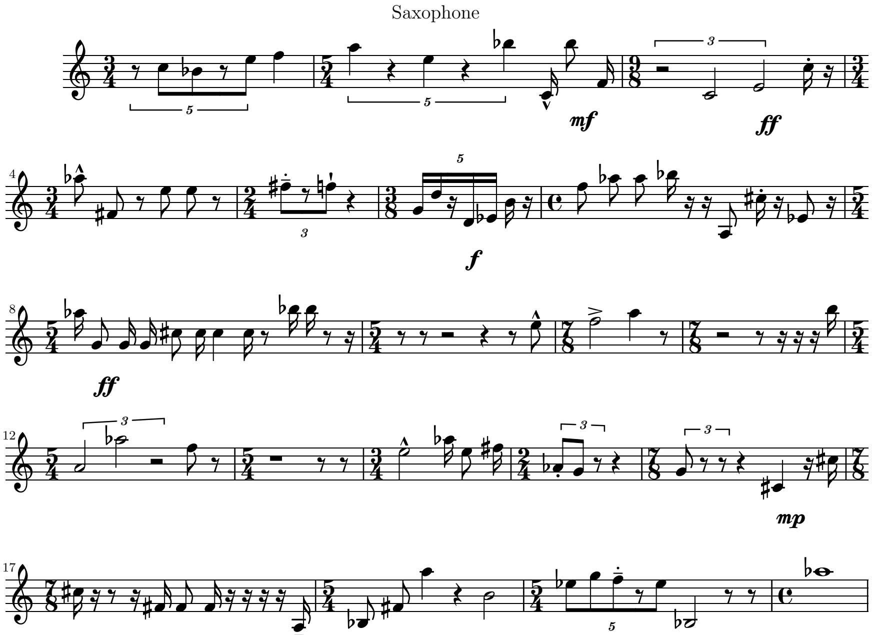
Kevin C. Baird

Saxophone



No Clergy

Kevin C. Baird

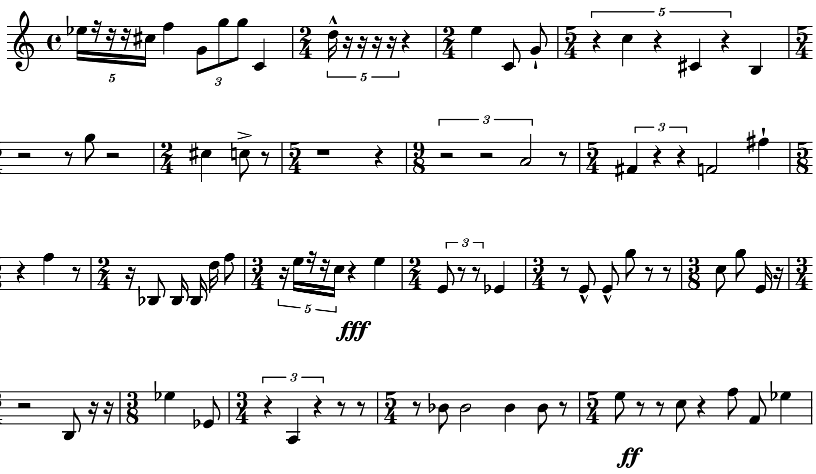


1

No Clergy

Kevin C. Baird

Saxophone

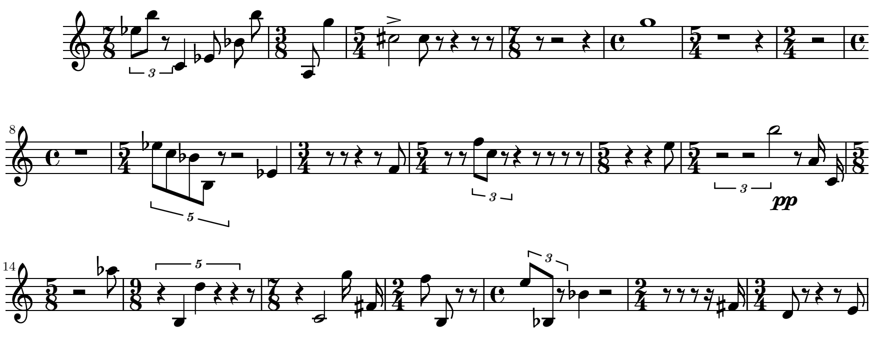


1

No Clergy

Kevin C. Baird

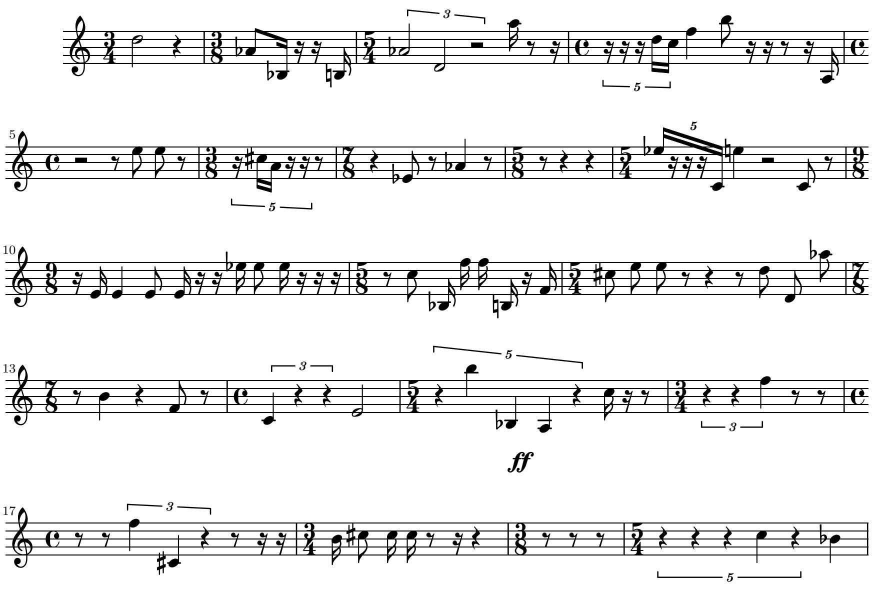
Saxophone



No Clergy

Kevin C. Baird

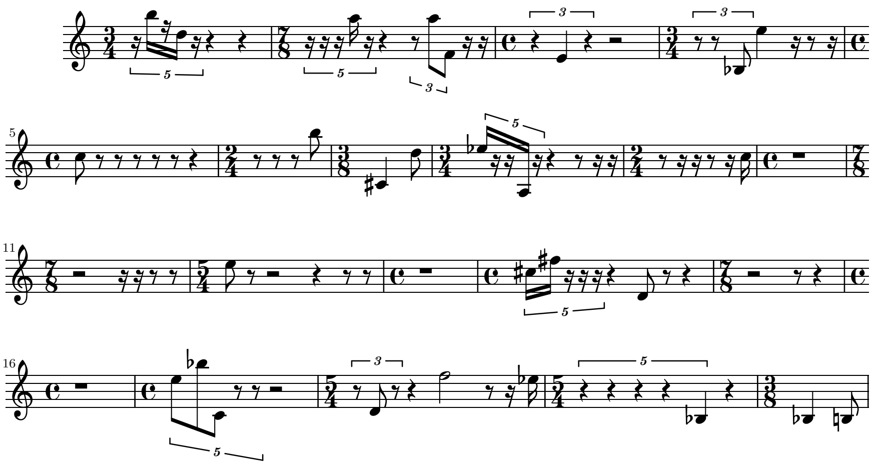
Saxophone



No Clergy

Kevin C. Baird

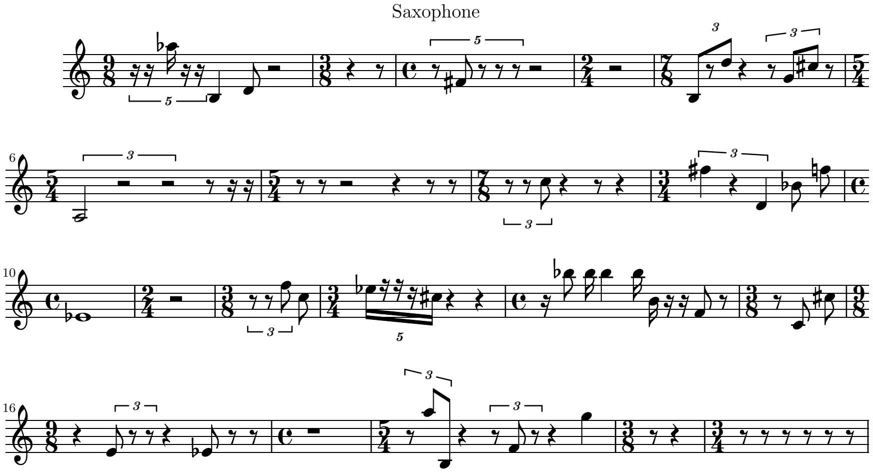
Saxophone



No Clergy

Kevin C. Baird

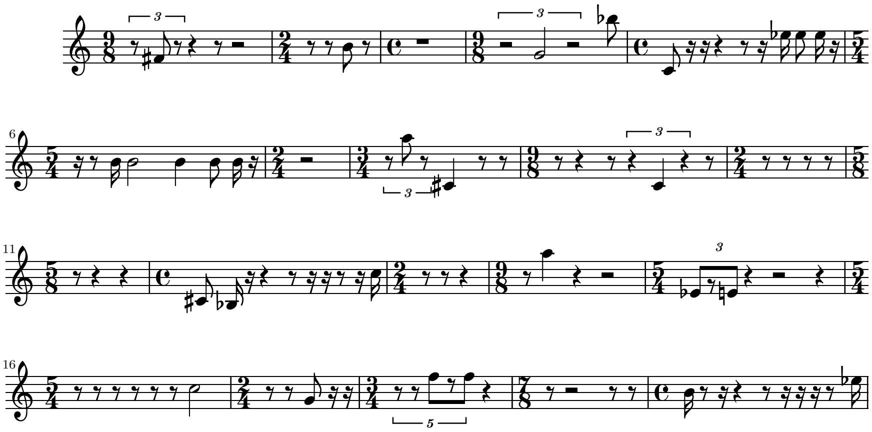
Saxophone



No Clergy

Kevin C. Baird

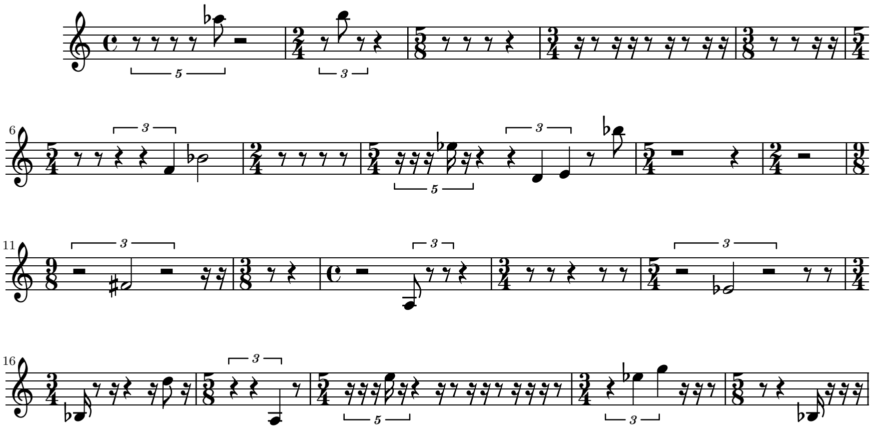
Saxophone



No Clergy

Kevin C. Baird

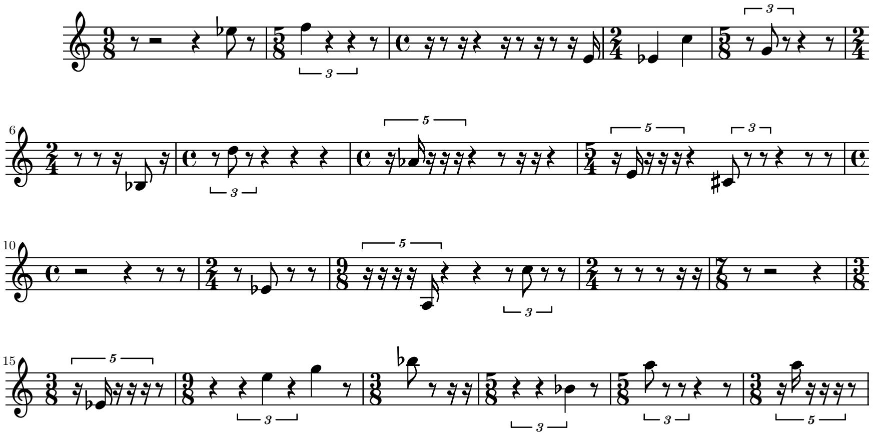
Saxophone



No Clergy

Kevin C. Baird

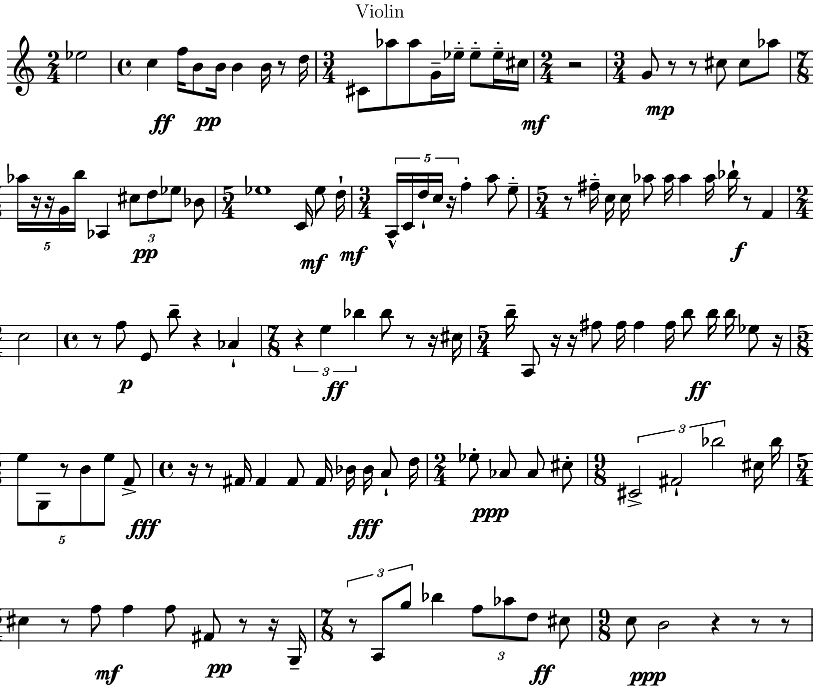
Saxophone



Chapter 5. Violin

No Clergy

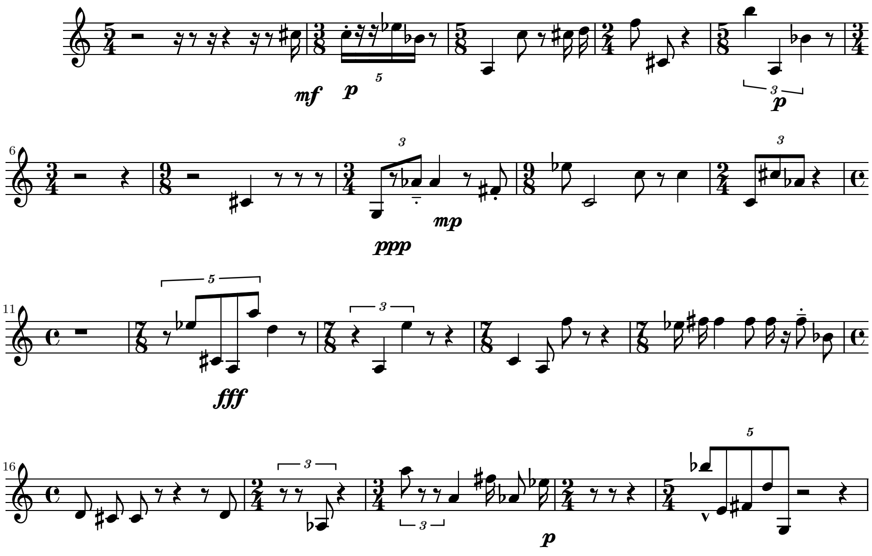
Kevin C. Baird



No Clergy

Kevin C. Baird

Violin

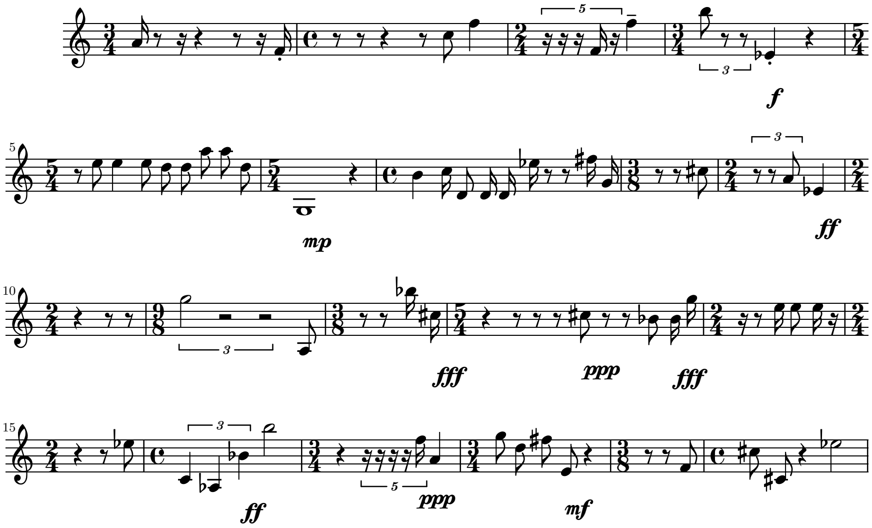


1

No Clergy

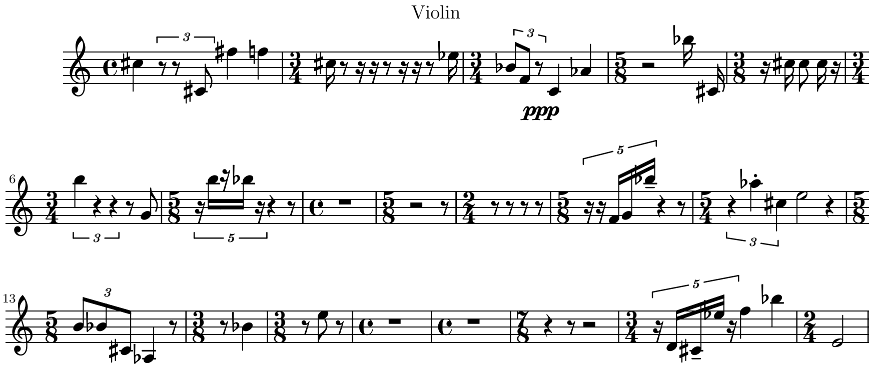
Kevin C. Baird

Violin



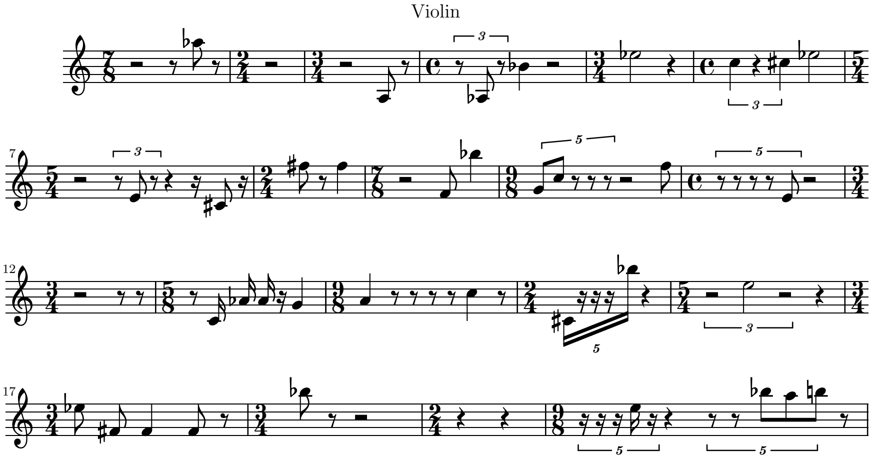
No Clergy

Kevin C. Baird



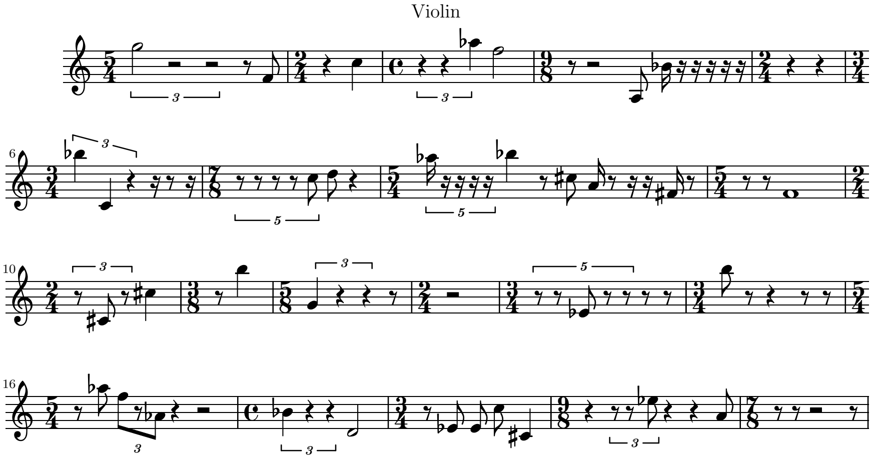
No Clergy

Kevin C. Baird



No Clergy

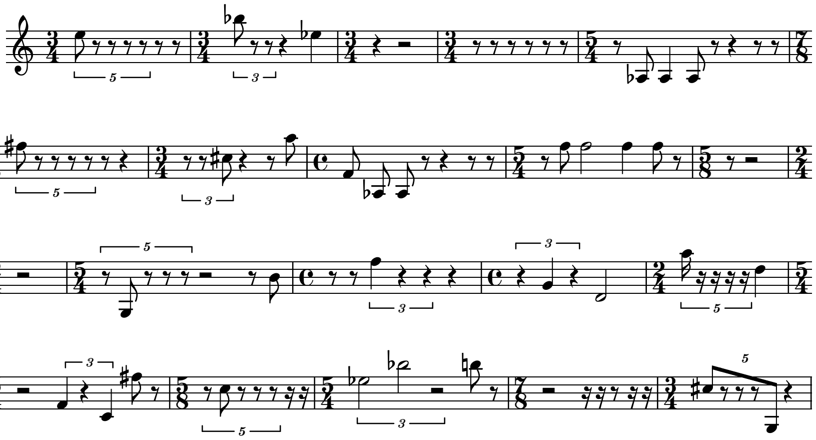
Kevin C. Baird



No Clergy

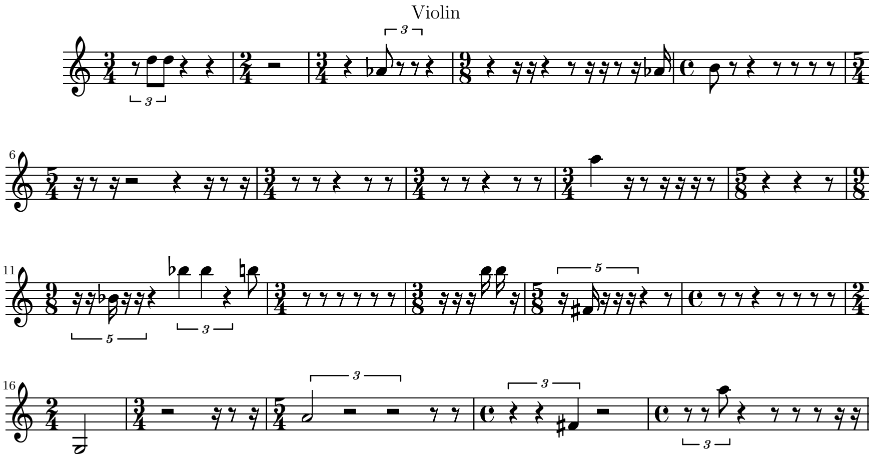
Kevin C. Baird

Violin



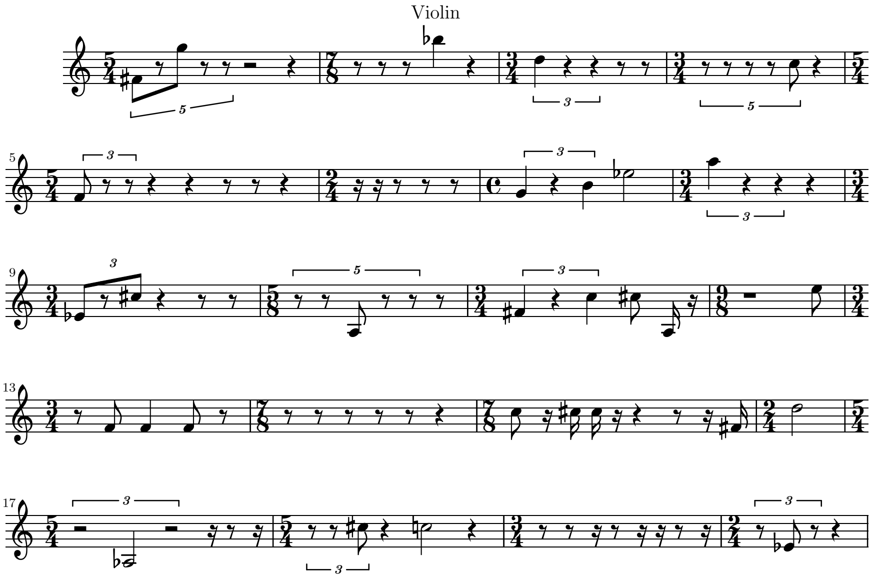
No Clergy

Kevin C. Baird



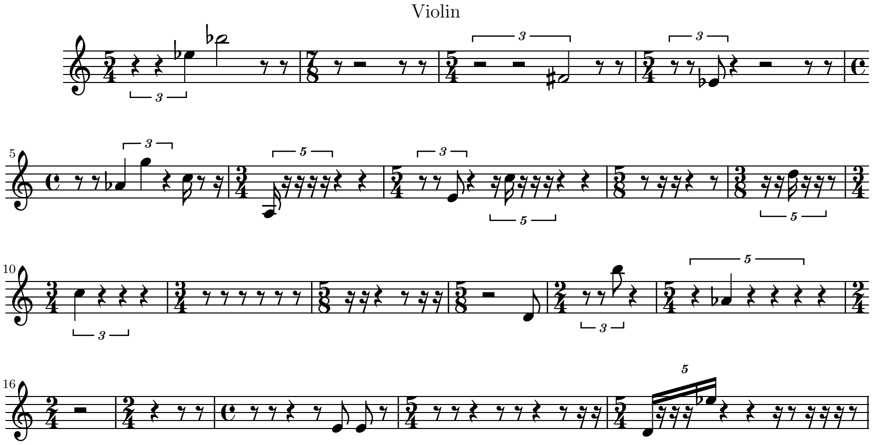
No Clergy

Kevin C. Baird



No Clergy

Kevin C. Baird



Part III. Python Modules

All Python Modules contain Class definitions and are located in `~/NoClergy/Python/`.

A note on Python variables. In my Python programs, I often use a convention whereby the last character of a variable's name is an upper-case letter representing the variable type. *I* signifies an integer, *S* signifies a string, *L* signifies a list, *D* signifies a dictionary (called a "hash" in some other languages), and *B* signifies a integer variable treated as a boolean. True booleans were added to Python after I started the project, and I saw no need to revise, given that ease of Debian package installation and backward compatibility were of greater value to me than avoiding integer variables which can be tested for truth value just as easily as a true boolean.

Table of Contents

Chapter 6. Config

```
#!/usr/bin/env python
# noclergy_config.py

# Copyright (C) 2004 Kevin C. Baird
#
# This file is part of ’No Clergy’.
#
# No Clergy is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

from noclergy_markov import Markov
import re

class Config:

  def __init__(self):
    self.instL = []
    self.instS = ”
  self.transpositionL = []
  self.modTupletI = 0
  self.modRestI = 0
  self.modDynI = 0
  self.modArtI = 0

def alter(self, valueS, ampI):
"""
Alter the variables used by the random generation of the score,
using the ampI value (-5 to 5) to indicate direction and amount
of the shift.
"""

  # ampI ranges from -5 to 5, excluding 0
  if ampI > 0:
    if valueS == ’rest’:
      mod = int((100 - self.restpcI) / (10 - ampI))
      self.restpcI += mod
    elif valueS == ’tuplet’:
      mod = int((100 - self.tupletpcI) / (10 - ampI))
      self.tupletpcI += mod
    elif valueS == ’art’:
      mod = int((100 - self.artpcI) / (10 - ampI))
      self.artpcI += mod
    elif valueS == ’dyn’:
      mod = int((100 - self.dynpcI) / (10 - ampI))
      self.dynpcI += mod
  elif ampI < 0:
    mod = (ampI + 10) * 0.1
    if valueS == ’rest’:
      self.restpcI *= mod
      self.restpcI = int(self.restpcI)
    elif valueS == ’tuplet’:
        self.tupletpcI *= mod
        self.tupletpcI = int(self.tupletpcI)
      elif valueS == ’art’:
        self.artpcI *= mod
        self.artpcI = int(self.artpcI)
      elif valueS == ’dyn’:
        self.dynpcI *= mod
        self.dynpcI = int(self.dynpcI)

  def readFeedback(self):
    """
    Use this to read user feedback used to alter pc variables.
    writeFile writes it out to the plain text file used by the Score Class.
    TODO: Read different feedback for each instrument.
    """
    self.feedbackFile = open(’/var/www/noclergy/feedback.html’, ’r’)
    readFeedbackB = 0
    for line in self.feedbackFile.readlines():
      #print "DEBUG:", line
      if re.search(’<!--begin-->’, line):
        readFeedbackB = 1
      elif re.search(’end 1 item’, line):
        readFeedbackB = 0
      if readFeedbackB:
        if re.search(’=’, line):
          if re.search(’tupletpc = .*’, line):
            self.modTupletS = re.findall(’tupletpc = .*’, line)[0]
            self.modTupletS = re.split(’= ’, line)[1]
            try:
              self.modTupletI = int(self.modTupletS)
            except:
              self.modTupletI = 0
          elif re.search(’restpc = .*’, line):
```

```
      self.modRestS = re.findall(’restpc = .*’, line)[0]
      self.modRestS = re.split(’= ’, line)[1]
      try:
       self.modRestI = int(self.modRestS)
      except:
       self.modRestI = 0
     elif re.search(’dynpc = .*’, line):
      self.modDynS = re.findall(’dynpc = .*’, line)[0]
      self.modDynS = re.split(’= ’, line)[1]
      try:
       self.modDynI = int(self.modDynS)
      except:
       self.modDynI = 0
     elif re.search(’artpc = .*’, line):
      self.modArtS = re.findall(’artpc = .*’, line)[0]
      self.modArtS = re.split(’= ’, line)[1]
      try:
       self.modArtI = int(self.modArtS)
      except:
       self.modArtI = 0
     elif re.search(’end 1 item’, line):
      readFeedbackB = 0
  self.feedbackFile.close()

 def readFile(self):
  """
  Reads pc variables from the plain text file, which have
  already been updated by readFeedback and the user data.
  TODO: Read different feedback for each instrument.
  """
  self.fileread = open(’NoClergy/config.txt’, ’r’)
  for line in self.fileread.readlines():
   if line[0] == ’#’:
        pass  # comment line, ignore
      elif re.search(’tupletpc = .*’, line):
        tupletS = re.findall(’tupletpc = .*’, line)[0]
        tupletS = re.split(’= ’, line)[1]
        self.tupletpcI = int(tupletS)
      elif re.search(’restpc = .*’, line):
        restS = re.findall(’restpc = .*’, line)[0]
        restS = re.split(’= ’, line)[1]
        self.restpcI = int(restS)
      elif re.search(’dynpc = .*’, line):
        dynS = re.findall(’dynpc = .*’, line)[0]
        dynS = re.split(’= ’, line)[1]
        self.dynpcI = int(dynS)
      elif re.search(’artpc = .*’, line):
        artS = re.findall(’artpc = .*’, line)[0]
        artS = re.split(’= ’, line)[1]
        self.artpcI = int(artS)
      elif re.search(’number_of_measures = .*’, line):
        mmS = re.findall(’number_of_measures = .*’, line)[0]
        mmS = re.split(’= ’, line)[1]
        self.numMeasuresI = int(mmS)
      elif re.search(’inst ’, line):
        instTransS = re.findall(’inst .*=.*’, line)[0]
        instS = re.split(’ ’, instTransS)[1]
        transS = re.split(’= ’, instTransS)[1]
        self.instL.append(instS)
        self.transpositionL.append(int(transS))
    self.fileread.close()

  def writeFile(self):
    """
    Writes pc variables into the working config file.
    TODO: Write different feedback for each instrument.
```

```
"""
self.filewrite = open('NoClergy/config.txt', 'w')
self.filewrite.write('# No Clergy config.txt, written by writeFile method\n')
self.filewrite.write('tupletpc = ' + str(self.tupletpcI) + '\n')
self.filewrite.write('restpc = ' + str(self.restpcI) + '\n')
self.filewrite.write('dynpc = ' + str(self.dynpcI) + '\n')
self.filewrite.write('artpc = ' + str(self.artpcI) + '\n')
self.filewrite.write('number_of_measures = ')
self.filewrite.write(str(self.numMeasuresI) + '\n')
try:
 for i in len(self.instL):
  outputS = 'inst ' + self.instL[i] + ' = '
  outputS += str(self.transpositionL)
  self.filewrite.write(outputS + '\n')
except:
 pass
self.filewrite.close()
```

Chapter 7. Header

```
#!/usr/bin/env python
# noclergy_header.py

# Copyright (C) 2004 Kevin C. Baird
#
# This file is part of ’No Clergy’.
#
# No Clergy is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307 USA

import os, random, re, sys, time
from noclergy_config import Config
from noclergy_markov import Markov
from xml.dom.ext.reader import Sax2

########################################
## BEGIN GLOBAL VARIABLE DECLARATIONS ##
########################################
```

```
config = Config()
config.readFile()
tupletpc = config.tupletpcI
restpc = config.restpcI
dynpc = config.dynpcI
artpc = config.artpcI
number_of_measures = config.numMeasuresI

### END GLOBAL VARIABLE DECLARATIONS ###

########################################
####### BEGIN CLASS DEFINITIONS #######
########################################

class Header:
  """
  Prints Lilypond boilerplate (version, header, etc.)
  identifying me and this piece No Clergy.
  """

  def __init__(self, config):
    self.tupletpcI = config.tupletpcI
    self.restpcI = config.restpcI
    self.artpcI = config.artpcI
    self.dynpcI = config.dynpcI

  def printout(self, inst):
    outputS = ”
    outputS += ’\\version "2.1.26"\n’
    outputS += ’\\include "english.ly"\n’
    outputS += ’\\header {\n’
    outputS += ’  title = "No Clergy"\n’
    self.subtitleS = ’  subtitle = "(’
    self.subtitleS += ’tupletpc=’ + repr(self.tupletpcI) + ’, ’
    self.subtitleS += ’restpc=’ + repr(self.restpcI) + ’, ’
    self.subtitleS += ’artpc=’ + repr(self.artpcI) + ’, ’
    self.subtitleS += ’dynpc=’ + repr(self.dynpcI)
    self.subtitleS += ’)"\n’
    # outputS += self.subtitleS
    # turn this back on if checking pc
    # variables is useful for debugging
    outputS += ’  instrument = ’
    if inst == ’sax’:
      outputS += ’"Saxophone"’
    elif inst == ’clar’:
      outputS += ’"Clarinet"’
    elif inst == ’vn’:
      outputS += ’"Violin"’
    outputS += ’\n’
    outputS += ’  composer = "Kevin C. Baird"\n’
    outputS += ’  tagline = "’
    outputS += ’Copyright (c) 2004, Kevin C. Baird, ’
    outputS += ’Released under the GNU General Public License ’
    outputS += ’(http://www.gnu.org)’
    outputS += ’"\n’
    outputS += ’}\n’
    outputS += ’##(set-global-staff-size 15)\n’
    return outputS

######## END CLASS DEFINITIONS ########
```

Chapter 8. Markov

```
#!/usr/bin/env python
# noclergy_markov.py

# Copyright (C) 2004 Kevin C. Baird
#
# This file is part of ’No Clergy’.
#
# No Clergy is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

import random

class Markov:
  """
  A set of operations dealing with Markov chains,
  organized into a single Object.
  """
```

```
def __init__(self):
  """
  Create empty dictionaries for each note attribute.
  """
  self.pcD = {}
  self.midi_pitchD = {}
  self.artD = {}
  self.dynD = {}
  self.durD = {}

def check(self, attr):
  """
  Double-check contents of the Markov Chain.
  """
  if attr == ’dur’:
    dict = self.durD
  if attr == ’dyn’:
    dict = self.dynD
  elif attr == ’art’:
    dict = self.artD
  elif attr == ’midi_pitch’:
    dict = self.midi_pitchD
  itemL = dict.items()
  # rewrite to accomodate 2nd order chain
  for item in itemL:
    (key, values) = item
    print key, ’=>’,
    for value in values:
      print value,
    print

def construct(self, attr, notesL):
  """
```

```
Takes in a list of notes and an attribute to construct a
Markov chain of, such as ’pc’ for pitch class.
"""
if attr == ’pc’:
  dict = self.pcD
elif attr == ’midi_pitch’:
  dict = self.midi_pitchD
elif attr == ’art’:
  dict = self.artD
elif attr == ’dur’:
  dict = self.durD
elif attr == ’dyn’:
  dict = self.dynD
for i in range(len(notesL)-1):
  note = notesL[i+1]
  previous_note = notesL[i]
  if attr == ’pc’:
    value = note.pc
    previous_value = previous_note.pc
  elif attr == ’midi_pitch’:
    value = note.midi_pitch
    previous_value = previous_note.midi_pitch
  elif attr == ’art’:
    value = note.art
    previous_value = previous_note.art
  elif attr == ’dyn’:
    value = note.dyn
    previous_value = previous_note.dyn
  elif attr == ’dur’:
    value = note.dur
    previous_value = previous_note.dur
  try:
    dict[previous_value].append(value)
```

```
   except:
    dict[previous_value] = []

 def construct2(self, attr, notesL):
  """
  Takes in a list of notes and an attribute to construct a
  2nd-order Markov chain of, such as ’pc’ for pitch class.
  """
  if attr == ’pc’:
   dict = self.pcD
  elif attr == ’midi_pitch’:
   dict = self.midi_pitchD
  elif attr == ’art’:
   dict = self.artD
  elif attr == ’dur’:
   dict = self.durD
  elif attr == ’dyn’:
   dict = self.dynD
  for i in range(len(notesL)-1):
   note = notesL[i+1]
   previous_note = notesL[i]
   try: prev2_note = notesL[i-1]
   except: prev2_note = Note(self.transposition)
   if attr == ’pc’:
    value = note.pc
    previous_value = previous_note.pc
    try: prev2_value = prev2_note.pc
    except: pass
   elif attr == ’midi_pitch’:
    value = note.midi_pitch
    previous_value = previous_note.midi_pitch
    try: prev2_value = prev2_note.midi_pitch
    except: pass
   elif attr == ’art’:
    value = note.art
    previous_value = previous_note.art
    try: prev2_value = prev2_note.art
    except: pass
   elif attr == ’dur’:
    value = note.dur
    previous_value = previous_note.dur
    try: prev2_value = prev2_note.dur
    except: pass
   elif attr == ’dyn’:
    value = note.dyn
    previous_value = previous_note.dyn
    try: prev2_value = prev2_note.dyn
    except: pass
   try:
    dict[prev2_value][previous_value].append(value)
   except:
    try:
     dict[prev2_value][previous_value] = []
     dict[prev2_value][previous_value].append(value)
    except:
     dict[prev2_value] = {}
     dict[prev2_value][previous_value] = []
     dict[prev2_value][previous_value].append(value)
    pass

 def extract(self, attr, value):
  """
  Takes in an attribute type and value, and outputs a ’next value’,
  determined by Markov Chain.
  """
  if attr == ’art’:
      dict = self.artD
    elif attr == 'dur':
      dict = self.durD
    elif attr == 'dyn':
      dict = self.dynD
    elif attr == 'midi_pitch':
      dict = self.midi_pitchD
    ## et cetera
    try:
      next_value = random.choice(dict[value])
    except:
      #next_value = ' % tried to read from ' + value + '\n'
      next_value = ' '
    return next_value

  def extract2(self, attr, previous_value, value):
    """
    Takes in an attribute type and value, and outputs a 'next value',
    determined by Markov Chain.
    """
    if attr == 'art':
      dict = self.artD
    elif attr == 'dur':
      dict = self.durD
    elif attr == 'dyn':
      dict = self.dynD
    elif attr == 'midi_pitch':
      dict = self.midi_pitchD
    ## et cetera
    #print 'DEBUG:\n', dict
    try:
      next_value = random.choice(dict[previous_value][value])
    except:
```

```
    if attr == ’midi_pitch’:
      next_value = 0
    else:
      next_value = ’ ’
  return next_value
```

Chapter 9. Measure

```
#!/usr/bin/env python
# noclergy_measure.py

# Copyright (C) 2004 Kevin C. Baird
#
# This file is part of ’No Clergy’.
#
# No Clergy is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

from noclergy_config import Config
from noclergy_markov import Markov
from noclergy_note import Note
from noclergy_object import Object
from xml.dom.ext.reader import Sax2
import random

class Measure(Object):
```

```
"""
Occurring within a Score,
this contains a list of Notes, a meter, etc.
"""
def __init__(self, num, transpose):
  self.notes = []
  self.filled = 0
  # how many 16ths in the measure have been used up?
  self.num = num # mm number, as commonly understood
  self.prev_dyn = ”
  self.tempo = 0
  self.transposition = transpose
  self.last_dur = 0
  self.length = 0 # how many 16ths in the measure?
  self.ticks = 0  # how many ticks in the measure?

def addNote(self, note):
  """
  Adds the Note argument to its list of Notes, ensuring that
  it will fit within the remaining beats in the Measure.
  """
  if note.ticks > self.remaining:
    while note.ticks > self.remaining:
      note.setDuration(’rand’, self.last_dur)
  # don’t delete the above, even if you think it’s redundant
  self.notes.append(note)
  self.filled += note.ticks
  self.remaining -= note.ticks

def allRestsInBeat(self):
  """
  Cycle through every note, if a beat consists entirely of
  rests, replace them with a single rest of a beat’s duration.
  """
  beatL = []
  filledI = 0
  some_non_rests = 0
  tempL = []
  for note in self.notes:
    if filledI % note.rvalues[self.bottom] == 0:
      downbeatB = 1
    else:
      downbeatB = 0
    if not note.pitch == ’r’:
      some_non_rests = 1
    filledI += note.dur
    beatL.append(note)
    if downbeatB:
      if some_non_rests:
        for sub_note in beatL:
          tempL.append(sub_note)
      else:
        temp_note = Note(self.transposition)
        temp_note.copy(note)
        temp_note.setDuration(note.rvalues[self.bottom], self.last_dur)
        tempL.append(temp_note)
      beatL = []

def assignXMLtuplet_nums(self):
  """
  This method steps through the notes in Measure.notes, checking for
  note.tuplet_type values other than 1. It assigns appropriate
  note.tuplet_num values so that Lilypond know when to close tuplet
  brackets.
  """
  tupnum = 0
```

```
for note in self.notes:
  if not note.tuplet:
    note.tuplet_num = 0
    note.tuplet_type = 1
    tupnum = 0
  else:
    tupnum += 1
    note.tuplet_num = tupnum

# BEGIN added since DOM
found_pitchB = 0
for note in self.notes:
  if note.tuplet:
    if not found_pitchB:
      if not note.pitch == ’r’:
        note.first_non_rest_tuplet = 1
        found_pitchB = 1
  else:
    found_pitchB = 0
self.notes.reverse()
found_pitchB = 0
for note in self.notes:
  if note.tuplet:
    if not found_pitchB:
      if not note.pitch == ’r’:
        note.last_non_rest_tuplet = 1
        found_pitchB = 1
  else:
    found_pitchB = 0
self.notes.reverse()
for note in self.notes:
  if note.first_non_rest_tuplet and note.last_non_rest_tuplet:
    note.first_non_rest_tuplet = 0
```

```
      note.last_non_rest_tuplet = 0
  # END added since DOM

def autoBeamSettings(self, max_separation=15):
  """
  Cycle through Notes, turn auto-beaming
  on or off as demanded aesthetically.
  """
  for i in range(len(self.notes)-1):
    prev_note = self.notes[i]
    current_note = self.notes[i+1]
    if prev_note.tuplet and prev_note.tuplet_num == 1:
      prev_note.autoBeamSuspendB = 1
    elif current_note.tuplet:
      if current_note.tuplet_num == current_note.tuplet_type:
        current_note.autobeamResumeB = 1
    elif current_note.tied:
      prev_note.autoBeamSuspendB = 1
      current_note.autobeamResumeB = 1
    if not prev_note.midi_pitch == 0:
      # if rather than elif, so that it will
      # break beams within tuplets as well
      if not current_note.midi_pitch == 0:
        separation = current_note.midi_pitch
        separation -= prev_note.midi_pitch
        separation = abs(separation)
        if separation >= max_separation:
          prev_note.autoBeamSuspendB = 1
          current_note.autobeamResumeB = 1
          # break beams if notes are
          # separated by 15+ semitones

def checkProperlyFilled(self):
  """
  Cycle through all the notes.
  Fails if they don’t add up to the total length.
  """
  temp_length = 0
  for note in self.notes:
    if note.tuplet and note.tuplet_type == note.tuplet_num:
      pass
    else:
      temp_length += note.dur
  if not temp_length == self.length:
    return 0
  else:
    return 1

def construct(self, tempo, method=’rand’):
  """
  Generates a Measure-full of musical data, mainly a list of Notes.
  """
  self.config = Config()
  self.config.tupletpcI = self.tupletpcI
  self.config.restpcI = self.restpcI
  self.config.artpcI = self.artpcI
  self.config.dynpcI = self.dynpcI
  self.config.numMeasuresI = self.numMeasuresI
  self.config.masterTupletL = self.masterTupletL
  self.config.tempTupletL = self.tempTupletL
  self.config.ticksI = self.ticksPerI
  self.config.instS = self.instS
  self.config.topL = self.topL
  self.previous_dynamics = ’ ’
  notesL = []
  self.setMeter(method)
```

```
 if self.num == 1:
  self.tempo = tempo
 self.debug_counter = 1
 while len(notesL) == 0:
 # outer loop makes sure there is at least 1 note in each measure
  while self.remaining:
   samplenote = Note(self.transposition)
   samplenote.addVariables(self.config)
   self.duration(samplenote)
   samplenote.setPitch(method)
   samplenote.setArticulation(method)
   samplenote.setDynamics(method)
   samplenote.remaining = self.remaining
   notesL.append(samplenote)
   self.addNote(samplenote)
 self.autoBeamSettings()

def constructMarkov(self, tempo, previous_note, prev2_note, markov):
 """
 Constructs a new Measure, defining
 Note characteristics via Markov methods.
 """
 self.config = Config()
 self.config.tupletpcI = self.tupletpcI
 self.config.restpcI = self.restpcI
 self.config.artpcI = self.artpcI
 self.config.dynpcI = self.dynpcI
 self.config.numMeasuresI = self.numMeasuresI
 self.config.masterTupletL = self.masterTupletL
 self.config.tempTupletL = self.tempTupletL
 self.config.ticksI = self.ticksPerI
 self.config.instS = self.instS
 self.config.topL = self.topL
```

```
  self.previous_dynamics = ’ ’
  if self.num == 1:
    self.tempo = tempo
  self.debug_counter = 1
  while len(self.notes) == 0:
  # outer loop makes sure there is at least 1 note in each measure
    while self.remaining:
      samplenote = Note(self.transposition)
      samplenote.addVariables(self.config)
      self.duration(samplenote)
      new_art = markov.extract2(’art’, prev2_note.art, previous_note.art)
      new_dur = markov.extract2(’dur’, prev2_note.dur, previous_note.dur)
      new_dyn = markov.extract2(’dyn’, prev2_note.dyn, previous_note.dyn)
      pr2_pitch = prev2_note.midi_pitch  # for wrapping in
      pr_pitch = previous_note.midi_pitch # printed output
      new_midi_pitch = markov.extract2(’midi_pitch’, pr2_pitch, pr_pitch)
      while random.randrange(100)+1 > self.restpcI and not new_midi_pitch:
         NMP = markov.extract2(’midi_pitch’, pr2_pitch, pr_pitch)
         new_midi_pitch = NMP # fixes wrapping in printed output
      new_midi_pitch = int(new_midi_pitch)
      samplenote.setPitch(new_midi_pitch)
      samplenote.setArticulation(new_art)
      samplenote.setDynamics(new_dyn)
      samplenote.remaining = self.remaining
      self.notes.append(samplenote)
      self.addNote(samplenote)
      # update previous values
      prev2_note = Note(self.transposition)
      prev2_note.copy(previous_note)
      previous_note = Note(self.transposition)
      previous_note.copy(samplenote)

def duration(self, samplenote):
   """
   This method attaches a duration (dur) to the Note argument. It
   reads the tupletpc variable from the main script for the percent
   change that a given note will be the start of a tuplet, subject to
   other limitations. The method automatically blocks durations that
   can not fit within the remaining measure.
   """
   samplenote.dur = self.ticks + 1 # dummy, force while loop
   while samplenote.dur >= self.remaining:
    samplenote.setDuration(’rand’, self.last_dur)
    # keep creating new notes until they
    # fit inside the remaining measure
   if random.randrange(100)+1 < self.tupletpcI and self.filled % 4 == 0:
    # only do a tuplet if the pc says to, only on a 1/4 downbeat (% 4),
    if samplenote.dur >= 4 and samplenote.ticks < self.remaining:
      # only subdivide 1/4 notes, only if it fits within the measure
      self.makeTuplets(samplenote)
   else:
    samplenote.setDuration(’rand’, self.last_dur)

  def filewrite(self, inst):
   """
   This method returns a string describing this Measure
   as a fragment of a MusicXML file.
   """
   XMLfileS = ”
   XMLfileS += ’ <measure number="’ + repr(self.num) + ’">\n’
   XMLfileS += ’  <attributes>\n’
   XMLfileS += ’   <divisions>4</divisions>\n’
   XMLfileS += ’   <key>\n’
   XMLfileS += ’  <fifths>0</fifths>\n’
   XMLfileS += ’   </key>\n’
   XMLfileS += ’   <time>\n’
```

```
    XMLfileS += ’  <beats>’ + repr(self.top) + ’</beats>\n’
    XMLfileS += ’  <beat-type>’ + repr(self.bottom) + ’</beat-type>\n’
    XMLfileS += ’   </time>\n’
    XMLfileS += ’   <clef>\n’
    XMLfileS += ’  <sign>G</sign>\n’
    XMLfileS += ’  <line>2</line>\n’
    XMLfileS += ’   </clef>\n’
    if not self.transposition == 0:
      XMLfileS += ’   <transpose>\n’
      XMLfileS += ’  <chromatic>’
      XMLfileS += str(self.transposition)
      XMLfileS += ’</chromatic>\n’
      XMLfileS += ’   </transpose>\n’
    XMLfileS += ’  </attributes>\n’
    if self.tempo > 0 and self.num == 1:
      # TODO: allow measures other than first?
      XMLfileS += self.tempoOut()
    for note in self.notes:
      XMLfileS += note.filewrite()
    XMLfileS += ’ </measure>\n’
    return XMLfileS

  def fillBeat(self, note, space_left_in_beat, note_list):
    """
    Fills space_left_in_beat with the longest note that will fit,
    reduces space_left_in_beat, and recurses.
    """
    temp_dur = 0
    local_space_left = space_left_in_beat
    while_note_list = []
    for i in range(20):
      while_note_list.append(note)
      # needs to be an item in a new list
```

```
   # like this to not overwrite old notes
  i = -1
  while local_space_left > 1:
   i += 1
   while_note_list[i] = Note(self.transposition)
   while_note_list[i].copy(note)
   temp_dur = note.get_nearest_2power_equal(local_space_left)
   while_note_list[i].setDuration(temp_dur, self.last_dur)
   note_list.append(while_note_list[i])
   local_space_left -= temp_dur
  if local_space_left == 1:
   note1 = Note(self.transposition)
   note1.copy(note)
   note1.setDuration(1, self.last_dur)
   note_list.append(note1)

 def fixTies(self):
  """
  Cycles through notes and prevents any note from being
  tied to a note of a different pitch. Also prevents the
  last note in a measure from being tied to anything.
  """
  for i in range(len(self.notes)-1):
   current_note = self.notes[i+1]
   prev_note = self.notes[i]
   if current_note.tuplet_type > 1:  # only check actual tuplets
    if current_note.tuplet_type == current_note.tuplet_num:
     current_note.tied = 0    # final tuplets can not be tied
   if not prev_note.pitch == current_note.pitch:
    prev_note.tied = 0
   elif not prev_note.octave == current_note.octave:
    prev_note.tied = 0
  self.notes.reverse()
```

```
    try: self.notes[0].tied = 0
    except: pass
    self.notes.reverse()

  def fromXML_DOM(self, measureFromXML):
    """
    Look for tags in measureFromXML.
    """
    for noteTag in measureFromXML.getElementsByTagName(’note’):
      noteFromXML = Note(self.transposition)
      noteFromXML.addVariables(self.config)
      noteFromXML.fromXML_DOM(noteTag)
      self.notes.append(noteFromXML)

  def makeDottedDisplay(self, rests=”):
    """
    Cycles through notes, if they are tied together and the first
    is twice as long, they are combined into one dotted note.
    """
    massageL = []
    each_durL = [1, 2, 4, 8]
    filledI = 0
    for i in range(len(self.notes)):
      each_note = self.notes[i]
      if filledI % each_note.rvalues[self.bottom] == 0:
        downbeatB = 1
      else:
        downbeatB = 0
      filledI += each_note.dur
      if not self.notes[i].tuplet:
        if not downbeatB:
          if rests == ’rests’ or not self.notes[i].pitch == ’r’:
            if self.notes[i].tied and self.notes[i-1].tied:
```

```
       if self.notes[i].pitch == self.notes[i-1].pitch:
        if self.notes[i-1].dur == 2 and self.notes[i].dur == 1:
          if self.notes[i].pitch == self.notes[i-1].pitch:
           self.notes[i].pitch = ’D’
           self.notes[i-1].dotted = 1
  for note in self.notes:
   if not note.pitch == ’D’:
    massageL.append(note)
  self.notes = []
  for note in massageL:
   self.notes.append(note)

 def makeTuplets(self, note):
  """
  This creates a set of tuplets (each of which is a normal Note). The type
  of tuplet is determined by the master_tupletL list in the main python
  script. That tuplet type then falls within the space of the closest
  lower power of 2: 5 in 4, 3 in 2, etc. In other words, it only creates
  tuplets which go faster than the base note type. If the tuplet created
  were to consist entirely of rests, it instead outputs a single rest of
  equivalent length.
  """
  self.all_rests = 1 # assume rests until you see a pitch in the tuplet
  tuplet_type = random.choice(self.masterTupletL)
  subdivide = note.get_nearest_2power(tuplet_type) # 3/2, 5/4, 7/4, etc.
  while self.all_rests: # keep looping until at least one tuplet is a pitch
   self.tupletL = []
   for i in range(1, tuplet_type+1):
    tupletnote = Note(self.transposition)
    tupletnote.addVariables(self.config)
    tupletnote.initTuplet(tuplet_type, i)
    tupletnote.setPitch(’rand’)
    if not tupletnote.pitch == ’r’:
```

```
    self.all_rests = 0
   tupletnote.setArticulation('rand')
   tupletnote.setDynamics('rand')
   tupletnote.setDuration(note.dur / subdivide, self.last_dur)
   tupletnote.setTicks(note.dur * self.ticksPerI / tuplet_type)
   self.tupletL.append(tupletnote)
 self.tupletBeaming()
 for tupletnote in self.tupletL:  # at least one tuplet is a note
  self.addNote(tupletnote)

def massageOutput(self):
 """
 Alters output to make it more legible to the performer -
 anything that's output-specific and not appropriate for storage:
 basically Lilypond-specific formatting commands
 auto-bracket tuplets
 add dots to durations as needed, etc.
 Constrasted with the 'massageStorage' method.
 """
 self.sortBeats()
 self.removeAdjacentStaccatos()
 self.removeAdjacentExpressions()
 #self.makeDottedDisplay()
 # pass 'rests' to also do rests
 self.fixTies()

def massageStorage(self):
 """
 Alters a Measure's stored musical data to make it
 more understandable. It does things that have meaning
 outside of Lilypond, such as combining adjacent rests
 (i.e. "r16 r16" becomes "r8", etc.)
```

```
    Things it won’t do:
    Converting single long notes which span beats into
    split notes with ties. (i.e. changing
    ’r16 a2 r8’ into ’r16 a8. ~ a4 ~ a16 r8’
    This complicates the storage mechanism without changing
    the musical events. The example above is an example of
    the sort of alteration that would be more appropriate
    for the ’massageOutput’ method.
    """
    self.allRestsInBeat()
    self.removeRestExpressions()

  def mutate_markov2(self, markov):
    """
    Do a Markov-based mutation at the Measure level.
    """
    for i in range(len(self.notes)):
      note = self.notes[i]
      #note = Note(self.transposition)
      previous_note = self.notes[i-1]
      if i > 1:
        prev2_note = self.notes[i-2]
      else:
        prev2_note = Note(self.transposition)
        prev2_note.art = ’ ’
        #prev2_note.dur = 4
        prev2_note.dyn = ’ ’
        prev2_note.midi_pitch = 0
      new_art = markov.extract2(’art’, prev2_note.art, previous_note.art)
      #new_dur = markov.extract2(’dur’, prev2_note.dur, previous_note.dur)
      new_dyn = markov.extract2(’dyn’, prev2_note.dyn, previous_note.dyn)
      pr2_pitch = prev2_note.midi_pitch  # for wrapping in
      pr_pitch = previous_note.midi_pitch # printed output
      new_midi_pitch = markov.extract2(’midi_pitch’, pr2_pitch, pr_pitch)
      while new_midi_pitch == 0 and random.randrange(100)+1 > restpc:
        new_midi_pitch = markov.extract2(’midi_pitch’, pr2_pitch, pr_pitch)
      try: new_midi_pitch = int(new_midi_pitch)
      except: print ’DEBUG: new_midi_pitch =’, new_midi_pitch
      note.setPitch(new_midi_pitch)
      #try: new_note.setDuration(int(new_dur))
      #except: new_note.setDuration(4)
      try: note.setArticulation(new_art)
      except: pass
      try: note.setDynamics(new_dyn)
      except: pass

  def printout(self, prev_dyn):
    """
    Prints an entire Measure in Lilypond format, complete with a meter
    declaration and a comment with the measure number within the piece.
    """
    self.prev_dyn = prev_dyn
    outputS = ’\n\n| % MEASURE ’ + str(self.num) + ’\n’
    meterS = ’\\time ’ + str(self.top) + ’/’ + str(self.bottom)
    outputS += meterS + ’ ’
    prev_midi_pitch = 0 # used to break beams separated by 12+ semitones
    for each_note in self.notes:
      outputS += each_note.output(self.prev_dyn, prev_midi_pitch) + ’ ’
      if not prev_dyn == ’ ’ and not each_note.pitch == ’r’:
        self.prev_dyn = each_note.dyn
      prev_midi_pitch = each_note.midi_pitch
    return outputS

  def removeAdjacentExpressions(self):
    """
    Cycles through Notes, removing expressions (articulations and
    dynamics) from all tied notes except the first. Used only for
    display in Lilypond.
    """
    prev_dyn = ’no previous value’
    prev_art = ’no previous value’
    newNoteL = []
    for i in range(len(self.notes)):
      note = self.notes[i]
      prev_note = self.notes[i-1]
      if prev_note.tied:
        note.dyn = ’ ’
        note.art = ’ ’
      newNoteL.append(note)
    self.notes = []
    for note in newNoteL:
      self.notes.append(note)

  def removeAdjacentStaccatos(self):
    """
    Cycles through note durations [1, 2, 4, 8]. For each of those
    durations, if it finds two adjacent notes with exactly the same
    pitch, dynamics, and articulations, it changes the subsequent
    notes into rests if the articulations are either ’staccato’ or
    ’staccatissimo’.
    """
    massageL = []
    each_durL = [1, 2, 4, 8]
    for each_dur in each_durL:
      filledI = 0
      for i in range(len(self.notes)):
        note = self.notes[i]
        p_note = self.notes[i-1]
        if not note.tuplet:
```

```
    if note.art in [’staccato’, ’staccatissimo’]:
     if note.art == p_note.art:
      if note.pitch == p_note.pitch or p_note.pitch == ’L’:
       if note.dyn == p_note.dyn:
        note.pitch = ’L’
        # ’L’ for later note
        # to be a rest
 for note in self.notes:
  if note.pitch == ’L’:
   note.setPitch(0)
  massageL.append(note)
 self.notes = []
 for note in massageL:
  self.notes.append(note)

def removeDuplicateNotes(self):
 """
 Eliminates Notes with pitch ’X’ from the Note list.
 """
 massageL = []
 for note in self.notes:
  if not note.pitch == ’X’:
   massageL.append(note)
 self.notes = []
 for note in massageL:
  self.notes.append(note)
def removeRestExpressions(self):
 """
 Goes through all the Notes in the Measure and removes any
 Articulations or Dynamic indicators if note.pitch is ’r’.
 """
 for note in self.notes:
  if note.pitch == ’r’:
       note.art = ’ ’
       note.dyn = ’ ’

def setMeter(self, top, bottom=8):
  """
  Assigns values to its own states determining meter: top and bottom.
  Meters range from 3/8 to 11/8, and are expressed as x/4 if possible.
  """
  if top == ’rand’:
    #using global topL list
    top = random.choice(self.topL)
    if top % 2 == 0:
      top = top/2
      bottom = 4
    else:
      bottom = 8
  self.top = top     # meter numerator
  self.bottom = bottom  # meter denominator
  self.ticks = (top * self.ticksPerI * 16)/bottom
  # how many ticks per measure?
  self.length = self.ticks/self.ticksPerI
  self.remaining = self.ticks

def sortBeats(self):
  """
  Re-arrange display (not real data) of notes and rests within a Measure
  to correspond to traditional notation practice. For example:
  replace "r8 ds2 r8" with "r8 ds8 ~ ds4 ~ ds8 r8"
  """
  backupNotesL = []
  # keep a backup of the original note list for robustness
  for note in self.notes:
    backupNotesL.append(note)
  newNotesL = []
  filledI = 0
  for note in self.notes:
    if filledI % note.rvalues[self.bottom] == 0:
      downbeatB = 1
    else:
      downbeatB = 0
    if note.tuplet_type > 1:
      newNotesL.append(note)
      if not note.tuplet_num == note.tuplet_type:
        filledI += note.dur
    elif downbeatB:
      newNotesL.append(note)
      filledI += note.dur
    else:
      self.spreadNotes(filledI, note, newNotesL)
      filledI += note.dur
  if filledI == self.length:  # only effects changes if there is no
    self.notes = []    # rhythmic error, i.e. too many or too
    for note in newNotesL:  # few notes in the measure
      self.notes.append(note)
  if not self.checkProperlyFilled():
    self.notes = []
    for note in backupNotesL:
      self.notes.append(note)

def spreadNotes(self, filledI, note, note_list):
  """
  Spreads out subnotes across the length of the input Note,
  breaking at the beat.
  """
  new_dur = 0
  original_dur = note.dur
```

```
    next_beat = (filledI/note.rvalues[self.bottom])+1
    next_beat *= note.rvalues[self.bottom]
    space_left_in_beat = next_beat - filledI
    if note.dur > space_left_in_beat and space_left_in_beat > 0:
      self.fillBeat(note, space_left_in_beat, note_list)
      self.fillBeat(note, original_dur-space_left_in_beat, note_list)
    else:
      note_list.append(note)

  def tempoOut(self):
    """
    Outputs tempo indications according to MusicXML spec.
    """
    outputS = ’  <direction><sound tempo="’
    outputS += repr(self.tempo)
    outputS += ’" /></direction>\n’
    return outputS

  def tupletBeaming(self):
    """
    Goes through all the notes/rests in a tuplet set
    and assigns note.first_non_rest_tuplet and
    note.last_non_rest_tuplet as appropriate.
    """
    note_yetB = 0
    for note in self.tupletL:
      if not note_yetB:
        if not note.pitch == ’r’:
          note.first_non_rest_tuplet = 1
          note_yetB = 1
    self.tupletL.reverse()
    note_yetB = 0
    for note in self.tupletL:
```

```
    if not note_yetB:
      if not note.pitch == ’r’:
        note.last_non_rest_tuplet = 1
        note_yetB = 1
  self.tupletL.reverse()
  for note in self.tupletL:
    if note.first_non_rest_tuplet and note.last_non_rest_tuplet:
      note.first_non_rest_tuplet = 0
      note.last_non_rest_tuplet = 0
      # don’t do beams unless there are 2+ notes per tuplet
```

Chapter 10. Note

```
#!/usr/bin/env python
# noclergy_note.py

# Copyright (C) 2004 Kevin C. Baird
#
# This file is part of ’No Clergy’.
#
# No Clergy is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

from noclergy_config import Config
from noclergy_object import Object
import random, re

class Note(Object):
  """
  This is an individual sound/silence event within a Measure. It has states for
  pitch, duration, articulation, and dynamics, and tuplet (1 if it is a part of
a tuplet, 0 otherwise), tuplet_type (1 for normal notes), and tuplet_num (what
number am I out of 3 or 5 or whatever?). self.pitch is ’r’ if it’s a rest, and
duration is the number of 16th notes in its normal (i.e. non-tuplet) note type,
rather than the note type itself. So an eighth note has a dur of 2, not 8.
"""

def __init__(self, transpose):
  self.art = ”
  self.autoBeamSuspendB = 0
  self.autoBeamResumeB = 0
  self.autoBeamS = ”
  self.dotted = 0
  self.dyn = ”
  self.dynL = [’ppp’, ’pp’, ’p’, ’mp’, ’ ’,
        ’mf’, ’f’, ’ff’, ’fff’]
  self.artL = [’staccatissimo’, ’staccato’, ’marcato’,
        ’accent’, ’ ’, ’portato’, ’tenuto’]
  self.pcL = [’c’, ’cs’, ’d’, ’ef’, ’e’, ’f’,
       ’fs’, ’g’, ’af’, ’a’, ’bf’, ’b’]
  self.midi_pitch = 0
  self.octave = ”
  self.durL = [1, 2, 2, 2, 2, 4, 4, 4, 4, 8, 8, 8, 8, 16, 16, 16, 16]
  # normally, 16ths only 1/4 as likely as others
  self.dur16L = [1, 1, 1, 2, 2, 4, 8, 8, 16]
  # after a 16th, 16ths more likely
  self.rvalues = {1:16, 2:8, 3:8, 4:4, 6:4, 8:2, 12:2, 16:1}
  self.filewrite_durtypeD = {
    16:’whole’, 12:’half’, 8:’half’, 6:’quarter’, 4:’quarter’,
     3: ’eighth’, 2:’eighth’, 1:’sixteenth’, ’ ’: ’quarter’}
  self.transposition = transpose
  self.tuplet = 0    # is this note a tuplet?
  self.tuplet_type = 1  # change to 3, 5, etc. if tuplet
  self.tuplet_num = 1   # 1st is 1, 2nd is 2... 5th is 5
```

```
    self.first_non_rest_tuplet = 0
    self.last_non_rest_tuplet = 0
    # which is not a rest (for beaming in Lilypond)
    self.tied = 0

  def copy(self, note):
    """Makes this Note a copy of the Note argument."""
    self.art = note.art
    self.autoBeamSuspendB = note.autoBeamSuspendB
    self.autoBeamResumeB = note.autoBeamResumeB
    self.autoBeamS = note.autoBeamS
    self.dotted = note.dotted
    self.dur = note.dur
    self.dyn = note.dyn
    self.first_non_rest_tuplet = note.first_non_rest_tuplet
    self.last_non_rest_tuplet = note.last_non_rest_tuplet
    self.midi_pitch = note.midi_pitch
    self.pitch = note.pitch
    self.octave = note.octave
    self.tuplet = note.tuplet
    self.tuplet_type = note.tuplet_type
    self.tuplet_num = note.tuplet_num
    self.ticksPerI = note.ticksPerI
    self.tupletpcI = note.tupletpcI
    self.restpcI = note.restpcI
    self.artpcI = note.artpcI
    self.dynpcI = note.dynpcI
    self.numMeasuresI = note.numMeasuresI
    self.instS = note.instS
    self.setRange()
    try:
      self.instS = note.instS
    except:
      pass
    self.masterTupletL = note.masterTupletL
    self.tempTupletL = note.tempTupletL
    self.topL = note.topL

  def getInstRangeMulti(self, midi_pitch):
    """
    Returns 0 if within range, -1 if too high, and 1 if too low.
    These values are used for multiplication outside of this Method.
    Multiply a Note’s output from this Method by 12 and add it to
    the Note’s midi_pitch to bring it back in the instrument’s range.
    """
    if midi_pitch > self.highest:
      return -1
    elif midi_pitch < self.lowest and not midi_pitch == 0:
      print ’found a too low note’
      return 1
    else:
      return 0

  def initTuplet(self, type, tuplet_num):
    """
    Makes the Note the first of a set of tuplets of length type.
    """
    self.tuplet = 1
    self.tuplet_type = type
    self.tuplet_num = tuplet_num

  def filewrite(self):
    """
    This method returns a string consisting of an entire <note>
    element compliant with the MusicXML DTD. It is called within
    the Measure object, which contains a list of Notes.
```

```
"""
XMLfileS = ”
XMLfileS += ’  <note>\n’
if self.pitch == ’r’:
 XMLfileS += ’   <rest/>\n’
else:
 self.pitch_typeS = ’pitch’
 XMLfileS += ’   <pitch>\n’
 if re.search("’", self.octave):
  self.octaveS = repr(len(re.findall("’", self.octave))+2)
 else:
  self.octaveS = repr(2-len(re.findall(",", self.octave)))
 self.alterS = ”
 XMLfileS += ’  <step>’ + self.pitch[0] + ’</step>\n’
 if len(self.pitch) > 1:
  if self.pitch[1] == ’f’:
   self.alterS += ’-1’
  elif self.pitch[1] == ’s’:
   self.alterS += ’1’
 if not self.alterS == ”:
  XMLfileS += ’  <alter>’ + self.alterS + ’</alter>\n’
 XMLfileS += ’  <octave>’ + self.octaveS + ’</octave>\n’
 XMLfileS += ’   </pitch>\n’
XMLfileS += ’   <duration>’
XMLfileS += repr(self.rvalues[self.dur])
XMLfileS += ’</duration>\n’
XMLfileS += ’   <type>’
XMLfileS += self.filewrite_durtypeD[self.dur]
XMLfileS += ’</type>\n’
if self.dotted:
 XMLfileS += ’   <dot/>\n’
if self.tuplet:
 XMLfileS += ’  <time-modification>\n’
```

```
  XMLfileS += ’  <actual-notes>’
  XMLfileS += repr(self.tuplet_type)
  XMLfileS += ’</actual-notes>\n’
  XMLfileS += ’  <normal-notes>’
  XMLfileS += repr(self.get_nearest_2power(self.tuplet_type))
  XMLfileS += ’</normal-notes>\n’
  XMLfileS += ’ </time-modification>\n’
if not self.pitch == ’r’:
  if not self.art == ’ ’ or not self.dyn == ’ ’:
    if not self.art == ” or not self.dyn == ”:
      self.articulationS = ’ ’
      self.technicalS = ’ ’
      XMLfileS += ’  <notations>\n’
      if self.tuplet:
        if self.tuplet_num == 1:
          XMLfileS += ’ <tuplet type="start"/>\n’
        elif self.tuplet_num == self.tuplet_type:
          XMLfileS += ’ <tuplet type="stop"/>\n’
      if self.art == ’<stopped />’:
        self.technicalS += self.art
      elif self.art == ’upbow’:
        self.technicalS += ’<up-bow />’
      elif self.art == ’downbow’:
        self.technicalS += ’<down-bow />’
      if self.dyn:
        if not self.dyn == ’ ’:
          XMLfileS += ’ <dynamics><’
          XMLfileS += self.dyn
          XMLfileS += ’/></dynamics>\n’
      if not self.technicalS == ’ ’:
        XMLfileS += ’  <technical>\n’
        XMLfileS += ’ ’ + self.technicalS + ’\n’
        XMLfileS += ’  </technical>\n’
```

```
          if self.art == ’accent’:
            normal_artB = 1
          elif self.art == ’staccato’:
            normal_artB = 1
          elif self.art == ’tenuto’:
            normal_artB = 1
          elif self.art == ’staccatissimo’:
            normal_artB = 1
          else:
            normal_artB = 0
          if normal_artB:
            self.articulationS += ’<’ + self.art + ’ />’
          elif self.art == ’marcato’:
            self.articulationS += ’<strong-accent />’
          elif self.art == ’portato’:
            self.articulationS += ’<detached-legato />’
          if not self.articulationS == ’ ’:
            XMLfileS += ’  <articulations>’
            XMLfileS += self.articulationS[1:]
            XMLfileS += ’</articulations>\n’
          XMLfileS += ’   </notations>\n’
    XMLfileS += ’  </note>\n’
    return XMLfileS

  def fromXML_DOM(self, noteTag):
    """Look for tags in noteTag."""
    art = ”
    XMLartD = {’strong-accent’: ’marcato’, ’detached-legato’: ’portato’,
          ’up-bow’: ’upbow’, ’down-bow’: ’downbow’}
    if noteTag.getElementsByTagName(’rest’):
      self.pitch = ’r’
      self.art = ’ ’
      self.dyn = ’ ’
```

```
for durTag in noteTag.getElementsByTagName(’duration’):
 dur = int(durTag.firstChild.data)
 self.dur = 16 / dur
try:
 for dynTag in noteTag.getElementsByTagName(’dynamics’):
  self.dyn = dynTag.firstChild.tagName
except:
 self.dyn = ’ ’
try:
 for artTag in noteTag.getElementsByTagName(’articulations’):
  art = artTag.firstChild.tagName
except:
 try:
  for techTag in noteTag.getElementsByTagName(’technical’):
   art = techTag.firstChild.tagName
 except:
  art = ’ ’
if XMLartD.has_key(art):
 art = XMLartD[art]
self.art += art
for pitchTag in noteTag.getElementsByTagName(’pitch’):
 for stepTag in pitchTag.getElementsByTagName(’step’):
  step = stepTag.firstChild.data
 try:
  alterTag = pitchTag.getElementsByTagName(’alter’)[0]
  alter = int(alterTag.firstChild.data)
 except:
  alter = 0 # note might be a natural
 for octaveTag in pitchTag.getElementsByTagName(’octave’):
  octave = int(octaveTag.firstChild.data)
 self.getXMLpitch(step, alter, octave)
try:
 for timeModTag in noteTag.getElementsByTagName(’time-modification’):
        actualNotesNode = timeModTag.getElementsByTagName(’actual-notes’)
        # 2-step, made a Node to fix wrapping in printed output
        for actualNotesTag in actualNotesNode:
          self.tuplet_type = int(actualNotesTag.firstChild.data)
          self.tuplet = 1
    except:
      pass
    try:
      for notationsTag in noteTag.getElementsByTagName(’notations’):
        pass
    except: # if no notations tag
      self.art = ’ ’

  def get_nearest_2power(self, i):
    """
    Returns the nearest power of 2 below the argument. Useful for
    determining how many non-tuplet notes a tuplet occurs within the space
    of. E.g: 3 triplets occur within the space of 2, and this method
    returns 2 when given 3, etc. It does not support more exotic
    tuplet types (such as 7 in the space of 8).
    """
    k = 1
    while k < i:
      k *= 2
    return k / 2

  def get_nearest_2power_equal(self, i):
    """
    As above, equal to arg is OK too.
    """
    k = 1
    while k <= i:
      k *= 2
    return k / 2

  def getXMLpitch(self, step, alter, octave):
    """
    Extracts step, alter and octave tags from
    XML and generates internal pitch.
    """
    midi_pitchD = {’c’:0, ’d’:2, ’e’:4, ’f’:5, ’g’:7, ’a’:9, ’b’:11}
    pitchout = ”
    stepS = str(step)
    pitchout += stepS
    if alter == -1:
      pitchout += ’f’
    elif alter == 1:
      pitchout += ’s’
    if octave > 2:
      octaveS = "’" * (octave - 2)
    elif octave < 2:
      octaveS = "," * (-(2 - octave))
    else:
      octaveS = ”
    midi_pitch = (octave * 12) + midi_pitchD[stepS] + alter
    self.midi_pitch = midi_pitch
    self.pitch = pitchout
    self.octave = octaveS

  def ly_pitch(self):
    """
    Prepares a Note’s pitch data for transposed Lilypond output. It is
    generalized enough to handle transposition values other than the ones I
    happen to be using. I’ve decided to store the musical data in the XML
    files in concert pitch and only worry about transposition for display.
    I currently determine transposition values from the instrument name and
    a Dictionary, but I should eventually write it the XML file.
    """
    if not self.pitch == ’r’:
      pitchI = self.midi_pitch - self.transposition
      pitch = self.pcL[pitchI % 12]
      octave = self.octave
      if self.transposition < 0:
       if (pitchI % 12) < abs(self.transposition):
        if len(octave) > 0:
         if octave[len(octave)-1] == ’,’:
          octave = octave[0:len(octave)-1]
         else:
          pass
      elif self.transposition > 0:
       if (pitchI % 12) < abs(self.transposition):
        if len(octave) > 0:
         if octave[len(octave)-1] == "’":
          octave = octave[0:len(octave)-1]
         else:
          pass
    else:
      pitch = self.pitch
      octave = self.octave
    return pitch, octave

  def octaveMark(self, midi_pitch):
    out = ”
    if not midi_pitch == 0:
      if midi_pitch > 59:
       out += "’"
      if midi_pitch > 71:
       out += "’"
      if midi_pitch < 47:
       out += ","
    return out

  def output(self, prev_dyn, prev_midi_pitch):
    """Prints Lilypond-compliant text for the Note."""
    outputS = ”
    self.rhythm = self.rvalues[self.dur]
    if not self.midi_pitch == 0:
      if self.midi_pitch < self.lowest:
       outputS += ’\n%too low\n’
    self.midi_pitch += self.getInstRangeMulti(self.midi_pitch)
    # keep notes within instrument range
    auto_beam_turned_off = 0
    if self.autoBeamSuspendB:
      if self.tuplet:
       outputS += ’\n\\autoBeamOff\n’
    if self.tuplet and self.tuplet_num == 1:
      # first tuplet of the set
      outputS += ’\n  ’
      #outputS += ’\\autoBeamOff\n’
      outputS += ’\\times ’
      outputS += repr(self.tuplet_type-1) + ’/’
      outputS += repr(self.tuplet_type)
      outputS += ’ { ’
    if self.transposition == 0:
      pitch = self.pitch
      octave = self.octave
    else:
      pitch, octave = self.ly_pitch()
    outputS += pitch + octave + str(self.rhythm)
    if self.dotted:
      outputS += ’.’
    if self.dur < 4:
      # no beams for 1/4 notes or longer
      if self.first_non_rest_tuplet:
        # first tuplet note which isn’t a rest
        outputS += ’[’
      elif self.last_non_rest_tuplet:
        # first tuplet note which isn’t a rest
        outputS += ’]’
    if not self.art == ’ ’ and not self.art == ”:
      if not self.pitch == ’r’:
      # don’t output articulations on rests
        outputS += ’-\\’ + self.art
    if not self.dyn == ’ ’ and not self.dyn == ”:
      if not self.pitch == ’r’:
      # don’t output dynamics on rests
        if not self.dyn == prev_dyn:
          outputS += ’-\\’ + self.dyn
        # keeps semantic dynamics on each note,
        # but doesn’t print repeated dynamics
    if self.tied and not self.pitch == ’r’:
      outputS += ’ ~’
    if self.tuplet_type > 1:
      if self.tuplet_num == self.tuplet_type:
      # last tuplet of the set, note or rest
        outputS += ’ }\n’
    #outputS += ’ %’ + str(self.midi_pitch) + ’\n’
    if self.autoBeamResumeB:
      outputS += ’\n\\autoBeamOn\n’
    return outputS

  def setArticulation(self, art):
    """Accepts literal values, ’rand’, ’shorter’, or ’longer’."""
    if art == ’rand’:
      if random.randrange(100)+1 < self.artpcI and not self.pitch == ’r’:
```

```
    art = random.choice(self.artL)
   else:
    art = ’ ’
  elif art == ’shorter’:
    try: art = self.artL[self.artL.index(self.art) - 1]
    except: art = self.art
  elif art == ’longer’:
    try: art = self.artL[self.artL.index(self.art) + 1]
    except: art = self.art
  self.art = art

 def setDuration(self, dur, last_dur=0):
  """Accepts literal values or ’rand’."""
  if dur == ’rand’:
   if last_dur == 1:
    dur = random.choice(self.dur16L)
   else:
    dur = random.choice(self.durL)
  self.dur = dur
  self.ticks = dur * self.ticksPerI / self.tuplet_type

 def setDynamics(self, dyn):
  """Accepts literal values, ’rand’, ’softer’, or ’louder’."""
  if dyn == ’rand’:
   if random.randrange(100)+1 < self.dynpcI and not self.pitch == ’r’:
    dyn = random.choice(self.dynL)
   else:
    dyn = ’ ’
  elif dyn == ’softer’:
   try: dyn = self.dynL[self.dynL.index(self.dyn) - 1]
   except: dyn = self.dyn
  elif dyn == ’louder’:
   try: dyn = self.dynL[self.dynL.index(self.dyn) + 1]
   except: dyn = self.dyn
  self.dyn = dyn

 def setPitch(self, pitch):
  """Accepts literal values or ’rand’."""
  self.setRange()
  if pitch == ’rand’:
   if random.randrange(100)+1 > self.restpcI:
    pitch = random.randrange(self.lowest, self.highest)
   else:
    pitch = 0
  if pitch == 0:
   self.midi_pitch = pitch
   self.pitch = ’r’
   self.octave = ”
  else:
   self.midi_pitch = pitch
   self.pitch = self.pcL[self.midi_pitch % 12]
   self.octave = self.octaveMark(self.midi_pitch+2)

 def setRange(self):
  """Accepts literal values or ’rand’."""
  if self.instS == ’sax’:
   self.lowest = 60
   self.highest = 88
  elif self.instS == ’clar’:  # TODO: double-check ranges
   self.lowest = 52
   self.highest = 89
  elif self.instS == ’vn’:   # TODO: double-check ranges
   self.lowest = 55
   self.highest = 88
  else:
   self.lowest = 60
```

```
      self.highest = 88

  def setTicks(self, ticks):
    """Accepts literal values only."""
    self.ticks = ticks
```

Chapter 11. Object

```
#!/usr/bin/env python
# noclergy_object.py

# Copyright (C) 2004 Kevin C. Baird
#
# This file is part of ’No Clergy’.
#
# No Clergy is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

class Object:
  """Superclass, contains Methods used by multiple objects."""

  def addVariables(self, config):
    """
    Reads ’global’ configuration variables from
    the Config ’wrapper’ Object.
    """
```

```
self.config = config
self.tupletpcI = config.tupletpcI
self.restpcI = config.restpcI
self.artpcI = config.artpcI
self.dynpcI = config.dynpcI
self.numMeasuresI = config.numMeasuresI
self.ticksPerI = config.ticksI
self.instS = config.instS
self.masterTupletL = config.masterTupletL
self.tempTupletL = config.tempTupletL
self.topL = config.topL
```

Chapter 12. Paper

```
#!/usr/bin/env python
# noclergy_paper.py

# Copyright (C) 2004 Kevin C. Baird
#
# This file is part of 'No Clergy'.
#
# No Clergy is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

class Paper:
  """
  This is basically just more boilerplate for Lilypond.
  It prints paper and font size header information.
  It also creates the \midi section to allow creation of
  a MIDI file. It gets passed Score.tempo.
  """
```

```
def printout(self, tempo):
  outputS = ”
  #outputS += ’##(set-global-staff-size 15)\n’
  outputS += ’\\paper {\n’
  outputS += ’  ##(paper-set-staff-size (* 15 pt))\n’
  outputS += ’  linewidth = 7.5 \\in\n’
  outputS += ’  indent = 0.5 \\in\n’
  outputS += ’}\n’
  outputS += ’\\midi {\n’
  outputS += ’  \\tempo 8=’ + repr(tempo) + ’\n’
  outputS += ’}\n’
  return outputS
```

Chapter 13. Piece

```
#!/usr/bin/env python
# noclergy_piece.py

# Copyright (C) 2004 Kevin C. Baird
#
# This file is part of ’No Clergy’.
#
# No Clergy is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

class Piece:
  """
  The entire piece. This object exists to hold states and methods for
  cross-fertilizing mutations between instruments.
  """

  def __init__(self, scoreL):
    self.notes = []
    self.pitches = []
    self.durations = []
    self.articulations = []
    self.dynamics = []
    self.scoreL = scoreL

  def get_all_notes(self):
    for score in self.scoreL:
      score.get_all_notes()
      self.notes.append(score.notes)
      self.pitches.append(score.pitches)
      self.durations.append(score.durations)
      self.articulations.append(score.articulations)
      self.dynamics.append(score.dynamics)

  def sieve_pitches(self, scoreL, pitch_list, vary_amp=3):
    for score in scoreL:
      for measure in score.measures:
        for note in measure.notes:
          if not note.pitch == ’r’:
            if not pitch_list.count(note.midi_pitch % 12) > 0:
              vary = random.randrange(-vary_amp, vary_amp-1)
              if vary == 0:
                vary += 1
              note.setPitch(note.midi_pitch + vary)

  def sort_notes_in_mm(self, scoreL):
    for score in scoreL:
      for measure in score.measures:
        measure.notes.sort()
```

Chapter 14. Score

```
#!/usr/bin/env python
# noclergy_score.py

# Copyright (C) 2004 Kevin C. Baird
#
# This file is part of ’No Clergy’.
#
# No Clergy is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307 USA

from noclergy_config import Config
from noclergy_markov import Markov
from noclergy_measure import Measure
from noclergy_note import Note
from noclergy_object import Object
from noclergy_paper import Paper
from xml.dom.ext.reader.Sax2 import FromXmlStream
import os, random, re, time
```

```
#######################
# BEGIN LOCAL VARIABLES
#######################

localDTDpathS = ’"file:///home/kbaird/NoClergy/MusicXML/DTDs/partwise.dtd"’
# local version of the MusicXML DTD, alter pathname
# based on one’s own username or DTD location

#######################
# END LOCAL VARIABLES
#######################

class Score(Object):
  """
  This and its methods manage everything within
  the ’\score’ brackets in a Lilypond file.
  """

  def __init__(self, transpose):
    self.measures = []
    self.notes = []
    self.pitches = []
    self.durations = []
    self.articulations = []
    self.dynamics = []
    measuresL = []
    self.measures_so_far = 0
    self.step = 0
    self.alter = 0
    self.octave = 0
    self.tempo = 0
    self.transposition = transpose
```

```
def construct(self, config):
  """Loop each measure and append to Score’s list of measures."""
  number_of_measures = self.numMeasuresI
  while self.measures_so_far < number_of_measures:
    measure = Measure(self.measures_so_far+1, self.transposition)
    measure.addVariables(config)
    measure.construct(self.tempo)
    while not measure.checkProperlyFilled():
      measure = Measure(self.measures_so_far+1, self.transposition)
      measure.addVariables(config)
      measure.construct(self.tempo)
    self.measures.append(measure)
    self.measures_so_far += 1

def debug_out(self):
  """Debugging output."""
  outputS = ”
  for measure in self.measures:
    outputS += ’% MEASURE #’
    outputS += str(measure.num)
    outputS += ’, meter = ’
    outputS += str(measure.top)
    outputS += ’/’
    outputS += str(measure.bottom)
    outputS += ’, length = ’
    outputS += str(measure.top*16/measure.bottom)
    outputS += ’\n% ’
    for note in measure.notes:
      outputS += note.pitch
      outputS += str(note.dur)
      outputS += ’ ’
    outputS += ’\n’
    return outputS

  def digit_fix(self, timeS):
    if len(timeS) == 1:
      timeS = ’0’ + timeS
    return timeS

  def fileread(self, inst, config):
    """
    Read most recent file ’./nc/yyyy_ddd/hh_mm_ss.xml’
    (written by self.filewrite), assign to XMLfile for processing.
    """
    #self.config = config
    self.instS = inst
    self.good_filesL = []
    self.dirS = ’lilypond/xml/’ + inst + ’/’
    self.filenameL = os.listdir(self.dirS)
    for fileS in self.filenameL:
      if fileS[-3:] == ’xml’:
        self.good_filesL.append(fileS)
    self.good_filesL.sort()
    self.filenameS = self.dirS + self.good_filesL.pop()
    self.XMLfile = open(self.filenameS, ’r’)
    XMLdoc = FromXmlStream(self.XMLfile)
    self.fromXML_DOM(XMLdoc)

  def filewrite(self, inst):
    """
    Write to file ’./nc/yyyy_MM_dd-hh_mm_ss.xml,
    where yyyy is time.localtime(time())[0], MM is [1],
    dd is [2], hh is [3], mm is [4], ss is [5]
    """
    for measure in self.measures:
```

```
 measure.removeDuplicateNotes()
self.filenameS = ’lilypond/xml/’ + inst + ’/’
self.timeL = time.localtime(time.time())
self.filenameS += repr(self.timeL[0]) + ’_’ # yyyy
self.filenameS += self.digit_fix(repr(self.timeL[1])) + ’_’ # MM
self.filenameS += self.digit_fix(repr(self.timeL[2])) + ’-’ # dd
self.filenameS += self.digit_fix(repr(self.timeL[3])) + ’_’ # hh
self.filenameS += self.digit_fix(repr(self.timeL[4])) + ’_’ # mm
self.filenameS += self.digit_fix(repr(self.timeL[5])) + ’.xml’ # ss
XMLfile = open(self.filenameS, ’w’)
XMLfile.write(’<?xml version="1.0" encoding="UTF-8" standalone="no"?>\n’)
XMLfile.write(’<!DOCTYPE score-partwise PUBLIC\n’)
XMLfile.write(’"-//Recordare//DTD MusicXML 1.0 Partwise//EN"\n’)
#XMLfile.write(’"http://www.musicxml.org/dtds/partwise.dtd">\n’)
# online non-local version of the MusicXML DTD
XMLfile.write(localDTDpathS + ’>\n’)
# local version of the MusicXML DTD, alter pathname
# based on one’s own username or DTD location
XMLfile.write(’<score-partwise>\n’)
XMLfile.write(’<identification>\n’)
XMLfile.write(’  <creator>Kevin C. Baird</creator>\n’)
XMLfile.write(’  <rights>Copyright (c) 2004 Kevin C. Baird, ’)
XMLfile.write(’released under the GNU GPL</rights>\n’)
XMLfile.write(’</identification>\n’)
XMLfile.write(’ <part-list>\n’)
XMLfile.write(’  <score-part id="P1">\n’)
XMLfile.write(’   <part-name>’ + inst + ’</part-name>\n’)
XMLfile.write(’  </score-part>\n’)
XMLfile.write(’ </part-list>\n’)
XMLfile.write(’ <part id="P1">\n’)
for measure in self.measures:
 XMLfile.write(measure.filewrite(inst))
XMLfile.write(’ </part>\n’)
```

```
    XMLfile.write(’</score-partwise>\n’)
    XMLfile.close()

  def fromXML_DOM(self, XMLfile):
    """
    Look for tags in XMLfile.
    """
    for instTag in XMLfile.getElementsByTagName(’part-name’):
      self.instS = instTag.firstChild.data
    for measureTag in XMLfile.getElementsByTagName(’measure’):
      measureTag.normalize()
      for chromaticTag in measureTag.getElementsByTagName(’chromatic’):
        self.transposition = int(chromaticTag.firstChild.data)
      for soundTag in measureTag.getElementsByTagName(’sound’):
        self.tempo = int(soundTag.getAttribute(’tempo’))
      for timeTag in measureTag.getElementsByTagName(’time’):
        for beatsTag in timeTag.getElementsByTagName(’beats’):
          top = int(beatsTag.firstChild.data)
        for beatTypeTag in timeTag.getElementsByTagName(’beat-type’):
          bottom = int(beatTypeTag.firstChild.data)
      mmNum = int(measureTag.getAttribute(’number’))
      measureFromXML = Measure(mmNum, self.transposition)
      measureFromXML.addVariables(self.config)
      measureFromXML.setMeter(top, bottom)
      measureFromXML.tempo = self.tempo
      measureFromXML.fromXML_DOM(measureTag)
      measureFromXML.assignXMLtuplet_nums()
      self.measures.append(measureFromXML)

  def get_all_notes(self):
    for measure in self.measures:
      # possibly descend to measure level
      for note in measure.notes:
```

```
   if not note.pitch == ’X’:  # another failsafe
    self.notes.append(note)
    self.pitches.append(note.pitch)
    self.durations.append(note.dur)
    self.articulations.append(note.art)
    self.dynamics.append(note.dyn)

def mutate_markov(self):
 """
 Alter stored musical data in ’XMLfile’ according to data
 found in ’feedback’, the file containing input taken from
 the audience during the performance. The specific alterations
 that they inspire may depend heavily on precedents such as
 Iannis Xenakis’ ideas in ’Formalized Music’.
 """
 debug_outS = ”
 markov = Markov()
 markov.construct(’art’, self.notes)
 markov.construct(’midi_pitch’, self.notes)
 markov.construct(’dyn’, self.notes)
 previous_note = Note(self.transposition)
 previous_note.art = ”
 for measure in self.measures:
  while not measure.checkProperlyFilled():
   measure = Measure(self.measures_so_far+1, self.transposition)
   measure.construct(self.tempo)
  for i in range(len(measure.notes)):
   note = measure.notes[i]
   if i > 0:
    previous_note = measure.notes[i-1]
   prev_pitch = previous_note.midi_pitch
   # 2-step, fix wrapping in printed output
   new_midi_pitch = markov.extract(’midi_pitch’, prev_pitch)
```

```
      while new_midi_pitch == 0 and random.randrange(100)+1 > restpc:
        new_midi_pitch = markov.extract(’midi_pitch’, prev_pitch)
      try:
        new_midi_pitch = int(new_midi_pitch)
      except:
        new_midi_pitch = 0
      if new_midi_pitch > 0:
        new_midi_pitch += 24
      note.setPitch(new_midi_pitch)
      if note.midi_pitch == 0:
        new_art = ’ ’
        new_dyn = ’ ’
      else:
        new_art = markov.extract(’art’, previous_note.art)
        new_dyn = markov.extract(’dyn’, previous_note.dyn)
      note.setArticulation(new_art)
      note.setDynamics(new_dyn)

def mutate_markov2(self, config):
  """2nd Order Markov chain."""
  debug_outS = ”
  markov = Markov()
  newMeasuresL = []
  markov.construct2(’art’, self.notes)
  markov.construct2(’midi_pitch’, self.notes)
  markov.construct2(’dur’, self.notes)
  markov.construct2(’dyn’, self.notes)
  previous_note = self.measures[19].notes[-1]
  previous_note.addVariables(config)
  try: prev2_note = self.measures[19].notes[-2]
  except: prev2_note = self.measures[18].notes[-1]
  prev2_note.addVariables(config)
  for measure in self.measures:
      new_measure = Measure(measure.num, self.transposition)
      new_measure.addVariables(config)
      new_measure.setMeter(measure.top, measure.bottom)
      p_note = previous_note # 2-step variables to fix
      p2_note = prev2_note   # wrapping in printed output
      new_measure.constructMarkov(self.tempo, p_note, p2_note, markov)
      while not new_measure.checkProperlyFilled():
        # generally too long if not preperly filled
        new_measure = Measure(measure.num, self.transposition)
        new_measure.addVariables(config)
        new_measure.setMeter(measure.top, measure.bottom)
        new_measure.construct(self.tempo)
      newMeasuresL.append(new_measure)
      #measure.mutate_markov2(markov)
    self.measures = []
    for new_measure in newMeasuresL:
      self.measures.append(new_measure)

  def printclose(self):
    """
    Outputs Lilypond ending boilerplate,
    including paper definitions and the MIDI block.
    """
    outputS = ’\n’
    outputS += ’} % end notes\n’
    paper = Paper()
    outputS += paper.printout(self.tempo)
    #outputS += ’} % end Voice\n’
    outputS += ’} % end score\n’
    return outputS

  def printopen(self, inst):
    """Outputs Lilypond starting boilerplate."""
```

```
outputS = ’’
outputS += ’\\score { \n’
outputS += ’\\notes { \n’
outputS += ’  \\stemBoth\n’

# BEGIN Dynamics spacing
outputS += "  \\override Voice.DynamicText "
outputS += "#’no-spacing-rods = ##t\n"
outputS += "  \\override Voice.DynamicText "
outputS += "#’X-extent = #’(-12 . 12)\n\n"

outputS += ’  \\set autoBeaming = ##t\n’

outputS += ’  \\override Staff.DynamicLineSpanner ’
outputS += ’#\’padding = #2.8\n’
outputS += ’  \\override Staff.TupletBracket ’
outputS += ’#\’padding = #1.8\n’
outputS += ’  \\set midiInstrument = ’
if inst == ’sax’:
  outputS += ’"soprano sax"’
elif inst == ’clar’:
  outputS += ’"clarinet"’
elif inst == ’vn’:
  outputS += ’"violin"’
outputS += ’\n’
outputS += ’  \n’
if self.tempo > 0:
  outputS += ’  \\override Score.MetronomeMark ’
  outputS += ’#\’padding = #5\n’
  outputS += ’  \\tempo 8=’ + repr(self.tempo) + ’\n’
outputS += ’  \\clef treble\n’
return outputS
```

```
def printout(self, config):
  """Score level."""
  outputS = ”
  prev_dyn = ’ ’
  if len(self.measures):
    for measure in self.measures:
      measure.removeDuplicateNotes()
      temp_measure = Measure(measure.num, measure.transposition)
      temp_measure.addVariables(config)
      temp_measure.setMeter(measure.top, measure.bottom)
      for note in measure.notes:
        temp_measure.notes.append(note)
      temp_measure.massageOutput()
      if temp_measure.checkProperlyFilled():
        output_measure = temp_measure
      else:
        output_measure = measure
      outputS += output_measure.printout(prev_dyn)
      prev_dyn = measure.prev_dyn
  return outputS


def setTempo(self, tempo):
  """
  Self-explanatory, Score level.
  Values are eighth notes per minute.
  """
  if tempo == ’rand’:
    self.tempo = random.randrange(40, 75)
  else:
    try:
      self.tempo = tempo
    except:
      self.tempo = int(tempo)
```

Score

Part IV. Python Scripts and Config Files

All Python scripts are located in `~/NoClergy/Python/` unless otherwise noted with an absolute pathname.

A note on Python variables. In my Python programs, I often use a convention whereby the last character of a variable's name is an upper-case letter representing the variable type. *I* signifies an integer, *S* signifies a string, *L* signifies a list, *D* signifies a dictionary (called a "hash" in some other languages), and *B* signifies a integer variable treated as a boolean. True booleans were added to Python after I started the project, and I saw no need to revise, given that ease of Debian package installation and backward compatibility were of greater value to me than avoiding integer variables which can be tested for truth value just as easily as a true boolean.

Table of Contents

Chapter 15. cleanup.py

```
#!/usr/bin/env python
"""cleanup.py"""

# Copyright (C) 2004 Kevin C. Baird
#
# This file is part of ’No Clergy’.
#
# No Clergy is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

import os, re, sys

inst = sys.argv[1]
argS = sys.argv[2]
listdirS = ’lilypond/xml/’ + inst + ’/’
bakdirS = ’lilypond/xml/bak/’ + inst + ’/’

if argS == ’ls’:
  dir_listL = os.listdir(listdirS)
```

```
 dir_listL.sort()
 dir_listL.pop() # removes ’DTDs’
 for item in dir_listL:
  print ’list item =’, listdirS, item
 last_itemS = dir_listL.pop()
 print ’last item =’, listdirS, last_itemS
 print

elif argS == ’lsxml’:
 dir_listL = os.listdir(listdirS)
 dir_listL.sort()
 dir_listL.pop() # don’t consider ’DTDs’
 for item in dir_listL:
  if item[-3:] == ’xml’:
   print ’list item =’, listdirS, item
 last_itemS = dir_listL.pop()
 print ’last item =’, listdirS, last_itemS
 print

elif argS == ’mv’:
 dir_listL = os.listdir(listdirS)
 dir_listL.sort()
 dir_listL.pop() # don’t consider ’DTDs’
 if len(dir_listL):
  last_itemS = dir_listL.pop()
  dir_listL.append(last_itemS)
  for item in dir_listL:
   if item[-3:] == ’xml’:
    if not item == last_itemS:
     commandS = ’mv ’ + listdirS + item
     commandS += ’ ’ + bakdirS
     os.popen2(commandS)
```

```
if os.listdir(bakdirS):
  dir_listL = os.listdir(bakdirS)
  any_XML_files = 0
  for item in dir_listL:
    if item[-3:] == ’xml’:
      any_XML_files = 1

if any_XML_files:
  os.popen2(’bzip2 ’ + bakdirS + ’*.xml’)
```

Chapter 16. make_ly.py

```
#!/usr/bin/env python
"""make_ly.py"""

# Copyright (C) 2004 Kevin C. Baird
#
# This file is part of 'No Clergy'.
#
# No Clergy is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

import fileinput, os, random, re, sys, time
from noclergy_config import Config
from noclergy_header import Header
from noclergy_score import Score
from noclergy_measure import Measure
from noclergy_note import Note
from xml.dom.ext.reader import Sax2

########################################
```

```
## BEGIN GLOBAL VARIABLE DECLARATIONS ##
#########################################

master_tupletL = []
# this is the tuplet list that gets used later in the program
temp_tupletL = [3, 3, 3, 5, 5]
# what types of non-power of 2 tuplets are allowed?
# repetitions give greater likelihood to repeated options, i.e.
# [3, 3, 3, 5, 5] means 60% of all tuplets will be triplets of
# some sort, and 40% of all tuplets will be fives of some sort
# (subject to other limitations specific to the placement of
# the tuplet set in the score)

ticks = 1
# how many timing ticks (a la MIDI) per 16th note?
while len(temp_tupletL):
  # tuplets start from a base 1/4 note value (ticks * 4),
  # and divide by their tuplet_type to get their ticks value
  tuplet_mod = temp_tupletL.pop()
  master_tupletL.append(tuplet_mod)
  ticks *= tuplet_mod

# list of meter numerators
topL = [3, 4, 4, 5, 6, 6, 7, 8, 8, 9, 10, 10]
# x/4 meters are twice as likely, due to repetition above

try:
  inst = sys.argv[1]
except:
  inst = "you didn’t specify an instrument with sys.argv[1]"

instD = {}
instD = {’sax’:-2, ’clar’:-2, ’vn’:0}
```

```
# Dictionary of instruments and their transposition values
# TODO: derive instruments from config file

### END GLOBAL VARIABLE DECLARATIONS ###

########################################
########### BEGIN MAIN BODY ###########
########################################

# BEGIN instance declarations
config = Config()
config.readFile()
config.masterTupletL = master_tupletL
config.tempTupletL = temp_tupletL
config.ticksI = ticks
config.instS = inst
config.topL = topL
header = Header(config)
score = Score(instD[inst])
score.addVariables(config)
# END instance declarations

print header.printout(inst)
score.setTempo('rand')
score.construct(config)
score.filewrite(inst)
print score.printopen(inst)
#print score.debug_out()
print score.printout(config)
print score.printclose()

############ END MAIN BODY ############
```

Chapter 17. mutate.py

```
#!/usr/bin/env python
"""mutate.py"""

# Copyright (C) 2004 Kevin C. Baird
#
# This file is part of 'No Clergy'.
#
# No Clergy is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

import fileinput, os, random, re, sys, time
from noclergy_config import Config
from noclergy_header import Header
from noclergy_score import Score
from noclergy_measure import Measure
from noclergy_note import Note
from noclergy_piece import Piece
from xml.dom.ext.reader import Sax2
```

```
##########################################
## BEGIN GLOBAL VARIABLE DECLARATIONS ##
##########################################

master_tupletL = []
# this is the tuplet list that gets used later in the program
temp_tupletL = [3, 3, 3, 5, 5]
# what types of non-power of 2 tuplets are allowed?
# repetitions give greater likelihood to repeated options, i.e.
# [3, 3, 3, 5, 5] means 60% of all tuplets will be triplets of
# some sort, and 40% of all tuplets will be fives of some sort
# (subject to other limitations specific to the placement of
# the tuplet set in the score)

# list of meter numerators
topL = [3, 4, 4, 5, 6, 6, 7, 8, 8, 9, 10, 10]
# x/4 meters are twice as likely, due to repetition above

ticks = 1
# how many timing ticks (a la MIDI) per 16th note?
while len(temp_tupletL):
  # tuplets start from a base 1/4 note value (ticks * 4),
  # and divide by their tuplet_type to get their ticks value
  tuplet_mod = temp_tupletL.pop()
  master_tupletL.append(tuplet_mod)
  ticks *= tuplet_mod

try:
  directory = sys.argv[1]
except:
  directory = ’lilypond/ly/’

c_dom_seventhL = [0, 4, 7, 10]
```

```
# the PCs in the C Dominant 7th Chord
c_major_scaleL = [0, 2, 4, 5, 7, 9, 11]
# the PCs in the C Major scale
c_major_triadL = [0, 4, 7]
# the PCs in the C Major scale

instD = {’sax’:-2, ’clar’:-2, ’vn’:0}
# Dictionary of instruments and their transposition values
# TODO: derive instruments from config file

### END GLOBAL VARIABLE DECLARATIONS ###

########################################
##### BEGIN FUNCTION DEFINITIONS ######
########################################

def debugProperlyFilledScore(scoreL, instL):
  for i in range(len(scoreL)):
    score = scoreL[i]
    inst = instL[i]
    for measure in score.measures:
      if not measure.checkProperlyFilled():
        errorS = ’: ’
        for note in measure.notes:
          errorS += str(note.pitch) + str(note.dur) + ’ ’
        errorS += ’mm length = ’ + str(measure.length) + ’ ’
        raise Exception, inst + ’ ’ + str(measure.num) + errorS

def increaseRests(config):
  restpcImod = (config.restpcI + 100) / 2
  if restpcImod > 2:
    restpcImod = 2
  config.restpcI += restpcImod
```

```
#####  END FUNCTION DEFINITIONS  ######

########################################
########### BEGIN MAIN BODY ###########
########################################

# BEGIN instance declarations
config = Config()
config.readFile()
config.masterTupletL = master_tupletL
config.tempTupletL = temp_tupletL
config.ticksI = ticks
config.topL = topL
header = Header(config)
clar_score = Score(instD[’clar’])
clar_score.addVariables(config)
sax_score = Score(instD[’sax’])
sax_score.addVariables(config)
vn_score = Score(instD[’vn’])
vn_score.addVariables(config)
# END instance declarations

clar_score.fileread(’clar’, config)
sax_score.fileread(’sax’, config)
vn_score.fileread(’vn’, config)

scoreL = [clar_score, sax_score, vn_score]
instL = [’clar’, ’sax’, ’vn’]
# replace all this with reading from config.txt

for score in scoreL:
  config.instS = score.instS
  for measure in score.measures:
    measure.addVariables(config)
    measure.removeDuplicateNotes()

piece = Piece(scoreL)
piece.get_all_notes()
#piece.sort_notes_in_mm(scoreL)
#piece.sieve_pitches(scoreL, c_dom_seventhL)
# optional 3rd arg of variance amp, default 3

for i in range(len(scoreL)):
  ly_file = open(directory + instL[i] + '.ly', 'w')
  ly_file.write(header.printout(instL[i]))
  current_score = scoreL[i]
  #current_score.mutate_markov()  # 1st order Markov Chain
  current_score.mutate_markov2(config)  # 2nd order Markov chain
  for measure in current_score.measures:
    for note in measure.notes:
      #note.setArticulation('shorter')
      #note.setArticulation('longer')
      #note.setDynamics('louder')
      #note.setDynamics('softer')
      if note.pitch == 'X':
        raise Exception
      pass
  #current_score.remove_rest_expressions()
  scoreL[i].filewrite(instL[i])
  ly_file.write(current_score.printopen(instL[i]))
  ly_file.write(current_score.printout(config))
  ly_file.write(current_score.printclose())
  ly_file.close()

############ END MAIN BODY ############
```

Chapter 18. mutate_config.py

```
#!/usr/bin/env python
"""mutate_config.py"""

# Copyright (C) 2004 Kevin C. Baird
#
# This file is part of ’No Clergy’.
#
# No Clergy is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

import fileinput, os, random, re, sys, time
from noclergy_config import Config

#########################################
########### BEGIN MAIN BODY ###########
#########################################

# BEGIN instance declarations
config = Config()
```

```
config.readFile()
config.readFeedback()
#increaseRests(config)  # this gradually reduces the note density
config.alter('rest', config.modRestI)
config.alter('art', config.modArtI)
config.alter('dyn', config.modDynI)
config.alter('tuplet', config.modTupletI)
config.writeFile()

############ END MAIN BODY ############
```

Chapter 19. ~/NoClergy/config.txt

```
# Initial configuration file for No Clergy,
# Kevin Baird’s Doctoral Dissertation
# piece at the University at Buffalo

# variable [space] = [space] [value]
tupletpc = 50
restpc = 25
dynpc = 25
artpc = 25
number_of_measures = 20

# inst [space] name [space] = [space] [transposition value]
inst clar = -2
inst sax = -2
inst vn = 0
```

Chapter 20. /var/www/noclergy/feedback.html

`feedback.html` is located in `/var/www/noclergy/feedback.html` with a symbolic link in `~/NoClergy/`.

```
<!DOCTYPE html
    PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head><title>No Clergy Feedback Form</title></head>
<body>
<h1><cite>No Clergy</cite> Feedback Form</h1>
<!--begin-->

<hr />
<p>
Return to the <strong>No Clergy</strong>
<a href="./">Audience Feedback Form</a>.
</p>

</body>
</html>
```

Feedback from the audience is written immediately after the

<!--begin-->

comment tag, most recent first, via the `feedback.cgi` script.

Chapter 21. /usr/lib/cgi-bin/feedback.cgi

```
#!/usr/bin/env python
"""feedback.cgi"""
# put me in /usr/lib/cgi-bin/ on a Debian system
# possibly elsewhere for other OSes

# Copyright (C) 2004 Kevin C. Baird
#
# This file is part of ’No Clergy’.
#
# No Clergy is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

baseURL = ’http://nibbler.med.buffalo.edu/noclergy/’

import cgi, re
```

```
form = cgi.FieldStorage()
formS = ’<pre>\n’
for field in form:
 formS += field + ’ = ’ + form[field].value + ’\n’
formS += ’</pre>\n’

feedbackFile = open(’/var/www/noclergy/feedback.html’, ’r’)
new_feedbackFileS = ”
for line in feedbackFile.readlines():
 new_feedbackFileS += line
 if re.search(’<!--begin-->’, line):
 new_feedbackFileS += formS

feedbackFile = open(’/var/www/noclergy/feedback.html’, ’w’)
feedbackFile.write(new_feedbackFileS)
feedbackFile.close()

print "Content-type: text/html\n"
print "<html>\n"
print "<head><title>feedback results</title></head>\n"
print "<h1>Thank you for your feedback</h1>\n"
print "<body>\n"
print ’<p>Return to the <strong>No Clergy</strong> ’
print ’<a href="’ + baseURL + ’">’
print ’Audience Feedback Form</a>.</p>’
print "</body>\n</html>\n"
```

Part V. Bash Shell Scripts

All shell scripts are located in `~/NoClergy/` unless otherwise noted with an absolute pathname.

Table of Contents

Chapter 22. ~/nc_backup.sh

```
#!/bin/bash
# nc_backup.sh

# Copyright (C) 2004 Kevin C. Baird
#
# This file is part of ’No Clergy’.
#
# No Clergy is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
#
# Kevin can be reached at
# kcbaird@world.oberlin.edu or
# http://kevinbaird.net

# flush old dvi, ps, pdf files and make sure to backup feedback.cgi
source ./NoClergy/flush_dvi_pdf_ps.sh
cp /usr/lib/cgi-bin/feedback.cgi ./

# make bzip2 compressed tar archives with dated filenames
```

```
tar -cvf nc_scripts$(date ’+%Y_%m_%d-%H_%M_%S’).tar ./*.sh ./*.cgi
echo ’compressing scripts’
bzip2 nc_scripts*.tar
rm feedback.cgi
tar -cvf nc_folder$(date ’+%Y_%m_%d-%H_%M_%S’).tar ./NoClergy/*
echo ’compressing ~/NoClergy/ folder’
bzip2 nc_folder*.tar
tar -cvf nc_web$(date ’+%Y_%m_%d-%H_%M_%S’).tar /var/www/noclergy/*
echo ’compressing web’
bzip2 nc_web*.tar
tar -cvf nc_dbk$(date ’+%Y_%m_%d-%H_%M_%S’).tar ./Diss_dbk/*
echo ’compressing DocBook’
bzip2 nc_dbk*.tar

# secure copy tarballs into diss_backups folder on other machine
# ’kirk’ is identified in /etc/hosts
scp nc_scripts*.tar.bz2 kirk:/home/kbaird/diss_backups/
scp nc_folder*.tar.bz2 kirk:/home/kbaird/diss_backups/
scp nc_web*.tar.bz2 kirk:/home/kbaird/diss_backups/
scp nc_dbk*.tar.bz2 kirk:/home/kbaird/diss_backups/

# store backups in the backups folder
mv nc_scripts*.tar.bz2 backups/
mv nc_folder*.tar.bz2 backups/
mv nc_web*.tar.bz2 backups/
mv nc_dbk*.tar.bz2 backups/
```

Chapter 23. ~/noclergy.sh

```
#!/bin/bash
# noclergy.sh

# Copyright (C) 2004 Kevin C. Baird
#
# This file is part of ’No Clergy’.
#
# No Clergy is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
#
# Kevin can be reached at
# kcbaird@world.oberlin.edu or
# http://kevinbaird.net

source NoClergy/mv_oldfiles.sh

# -O flag to Python optimizes the module byte code files
python -O NoClergy/Python/mutate_config.py
python -O NoClergy/Python/mutate.py ’lilypond/ly/’
```

```
# add other lilypond source files for different instrumentation
lilypond --png -o lilypond/out/ lilypond/ly/clar.ly >>stdout.txt 2>>stderr.txt
lilypond --png -o lilypond/out/ lilypond/ly/sax.ly >>stdout.txt 2>>stderr.txt
lilypond --png -o lilypond/out/ lilypond/ly/vn.ly >>stdout.txt 2>>stderr.txt

# add other arguments and folders for different instrumentation
python NoClergy/Python/cleanup.py clar mv
python NoClergy/Python/cleanup.py sax mv
python NoClergy/Python/cleanup.py vn mv

source NoClergy/mv_output.sh
```

Chapter 24. ~/setup.sh

```
#!/bin/bash
# setup.sh

# Copyright (C) 2004 Kevin C. Baird
#
# This file is part of ’No Clergy’.
#
# No Clergy is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
#
# Kevin can be reached at
# kcbaird@world.oberlin.edu or
# http://kevinbaird.net

. NoClergy/mv_oldfiles.sh 2>/dev/null
. NoClergy/flush_feedback.sh 2>/dev/null
cp NoClergy/config.txt.orig NoClergy/config.txt

# add other arguments and folders for different instrumentation
# -O flag to Python optimizes the module byte code files
python -O NoClergy/Python/make_ly.py clar > lilypond/ly/clar.ly
python -O NoClergy/Python/make_ly.py sax > lilypond/ly/sax.ly
python -O NoClergy/Python/make_ly.py vn > lilypond/ly/vn.ly

# add other lilypond source files for different instrumentation
lilypond --png -o lilypond/out/ lilypond/ly/clar.ly >>stdout.txt 2>>stderr.txt
lilypond --png -o lilypond/out/ lilypond/ly/sax.ly >>stdout.txt 2>>stderr.txt
lilypond --png -o lilypond/out/ lilypond/ly/vn.ly >>stdout.txt 2>>stderr.txt

# add other arguments and folders for different instrumentation
python NoClergy/Python/cleanup.py clar mv
python NoClergy/Python/cleanup.py sax mv
python NoClergy/Python/cleanup.py vn mv

. NoClergy/mv_output.sh
```

Chapter 25. flush_dvi_pdf_ps.sh

```
#! /bin/bash
# flush_dvi_pdf_ps.sh

# Copyright (C) 2004 Kevin C. Baird
#
# This file is part of ’No Clergy’.
#
# No Clergy is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
#
# Kevin can be reached at
# kcbaird@world.oberlin.edu or
# http://kevinbaird.net

rm ~/lilypond/out/dvi/*.dvi 2> /dev/null
rm ~/lilypond/out/pdf/*.pdf 2> /dev/null
rm ~/lilypond/out/ps/*.ps 2> /dev/null
```

Chapter 26. flush_feedback.sh

```
#! /bin/bash
# flush_feedback.sh

# Copyright (C) 2004 Kevin C. Baird
#
# This file is part of ’No Clergy’.
#
# No Clergy is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
#
# Kevin can be reached at
# kcbaird@world.oberlin.edu or
# http://kevinbaird.net

rm /var/www/noclergy/feedback.html 2> /dev/null
cp NoClergy/feedback.html.orig /var/www/noclergy/feedback.html
chmod a+w /var/www/noclergy/feedback.html
# makes the feedback file writable by everyone.
# Otherwise, how could they enter any feedback?
```

Chapter 27. mv_oldfiles.sh

```
#!/bin/bash
# mv_oldfiles.sh

# Copyright (C) 2004 Kevin C. Baird
#
# This file is part of 'No Clergy'.
#
# No Clergy is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
#
# Kevin can be reached at
# kcbaird@world.oberlin.edu or
# http://kevinbaird.net

rm lilypond/ly/*-old.* 2> /dev/null
rm lilypond/out/pdf/*-old.* 2> /dev/null
rm lilypond/out/png/*-old.* 2> /dev/null

mv lilypond/ly/clar.ly lilypond/ly/clar-old.ly
mv lilypond/ly/sax.ly lilypond/ly/sax-old.ly
mv lilypond/ly/vn.ly lilypond/ly/vn-old.ly
mv lilypond/out/pdf/clar.pdf lilypond/out/pdf/clar-old.pdf
mv lilypond/out/pdf/sax.pdf lilypond/out/pdf/sax-old.pdf
mv lilypond/out/pdf/vn.pdf lilypond/out/pdf/vn-old.pdf

cp lilypond/out/png/clar-page1.png lilypond/out/png/clar-page1-old.png
cp lilypond/out/png/sax-page1.png lilypond/out/png/sax-page1-old.png
cp lilypond/out/png/vn-page1.png lilypond/out/png/vn-page1-old.png
# cp instead of mv, so the player isn’t stuck with no image file
```

Chapter 28. mv_output.sh

```
#!/bin/bash
# mv_output.sh

# Copyright (C) 2004 Kevin C. Baird
#
# This file is part of ’No Clergy’.
#
# No Clergy is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
#
# Kevin can be reached at
# kcbaird@world.oberlin.edu or
# http://kevinbaird.net

crop() {
  for file in $@; do
    convert -crop 744x800+0+0 $file cropped.png
    mv cropped.png $file
  done
}

mv lilypond/out/*.dvi lilypond/out/dvi/
mv lilypond/out/*.midi lilypond/out/midi/
mv lilypond/out/*.pdf lilypond/out/pdf/
mv lilypond/out/*.png lilypond/out/png/
mv lilypond/out/*.ps lilypond/out/ps/

crop lilypond/out/png/*.png

chmod 744 lilypond/out/png/*.png
# make images readable to the outside world
```

Bibliography

Books

Lejaren A Hiller, Jr. and Leonard M Isaacson. Copyright © 1959 McGraw-Hill Book Company, Inc.. 0-313-22158-8. McGraw-Hill. *Experimental Music*.

Magnus Lie Hetland. Copyright © 2002 Magnus Lie Hetland. 1-59059-00606. Apress. *Practical Python*.

Christopher A. Jones and Fred Drake, Jr. Copyright © 2002 O'Reilly & Associates, Inc. 0-596-00128-2. O'Reilly & Associates, Inc. *Python & XML*. XML Processing with Python.

Mark Lutz. Copyright © 2001, 1996 O'Reilly & Associates, Inc. 0-596-00085-5. O'Reilly & Associates, Inc. *Programming Python*. Solutions for Python Programmers.

Iannis Xenakis. Copyright © 1992 Pendragon Press. 1-57647-079-2. Pendragon Press. *Formalized Music, revised edition*. Thought and Mathematics in Music.

Musical Compositions

Earle Brown. Copyright © 1962. Associated Music Publishers, Inc. *Available Forms I and II*.

Lejaren A Hiller, Jr. and Leonard M Isaacson. Copyright © 1957. 0-313-22158-8. New Music Edition. *Iliac Suite for String Quartet*. 30. 3.

Software

The Apache Software Foundation. *Apache HTTP Server [http://httpd.apache.org]* .

The Debian Project. *Debian GNU/Linux [http://debian.org]* .

Han-Wen Nienhuys and Jan Nieuwenhuizen. *GNU Lilypond [http://lilypond.org]* .

Chet Ramey. *GNU Bash [http://www.gnu.org/software/bash/bash.html]* .

Guido van Rossum. *Python [http://python.org]* .

Appendix A. GNU General Public License

Version 2, June 1991

Free Software Foundation, Inc.
59 Temple Place, Suite 330
Boston
MA
02111-1307
USA

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software - to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps:

1. copyright the software, and
2. offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

Section 0

This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a “work based on the Program ” means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term “modification ”.) Each licensee is addressed as “you”.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

Section 1

You may copy and distribute verbatim copies of the Program’s source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

Section 2

You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

1. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
2. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
3. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License.

Exception:

If the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

Section 3

You may copy and distribute the Program (or a work based on it, under Section 2 in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

1. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

2. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

3. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

Section 4

You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

Section 5

You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

Section 6

Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

Section 7

If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

Section 8

If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

Section 9

The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

Section 10

If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY Section 11

BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

Section 12

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program’s name and a brief idea of what it does.> Copyright (C) <year> <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’. This is free software, and you are welcome to redistribute it under certain conditions; type ‘show c’ for details.

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than ‘show w’ and ‘show c’; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program ‘Gnomovision’ (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.