# MUMT 621: The music21 library

Ryan Groves
ryan.groves@mail.mcgill.ca

25 March 2013

## 1 Introduction

music21 is asset to musicologists in the 21st century. It was created by musicologists, and built for anyone who is curious about music and has a fundamental grasp of computer programming concepts. The suite of Python libraries provides a large set of musical objects on which many musical and programmatic operations can be performed. It defines these objects in an intuitive, hierarchical way, and is ideal for problems involving the investigation of event-based symbolic music analysis.

## 2 History

Music programming languages have been around for decades. Indeed, Max Mathews is widely known as the founder of music programming, creating the first widely-used music programming language MUSIC I in 1957 (Geiger 2005). He was employed with Bell laboratories at the time, and his software ran on the IBM 704 computer (Geiger 2005). This began a long series of MUSIC-N programming languages, most of which were developed at Bell laboratories. Mathews later invented the MUSIC V programming language, implemented in Fortran. The programming paradigms used in MUSIC V would later shape the methodology used for many music programming languages (Geiger 2005).

Another prominent figure in the music programming language field was Miller Puckette (Geiger 2005). Miller Puckette developed a graphical system which allowed users to define programming functionality using a series of visual objects and connections, named after Max Mathews (entitled MAX). It was developed at the Institut de Recherche et Coordination Acoustique/Musique (IRCAM), and built for the Apple Macintosh platform. Miller Puckette later developed a software package similar to MAX that was also open-source (unlike MAX), named Pure Data. Since then, there have been even more musical programming languages developed, for example SuperCollider and ChucK.

Even with the advance of music programming languages, there was still a need for an effective tool to catalyze the study and theoretical analysis of music symbolic scores.

The invention of the musical programming software Humdrum aimed to serve that need. Humdrum was created by David Huron (1997), and was the most widely adopted software package for the analysis of music symbolic data (Cuthbert and Ariza 2010) as of 2010. However, the process of combining different packages together to perform simple musicological tasks, as well as Humdrum's lack of an object-oriented structure restricted its use to only those who could tackle the programmatic challenges it posed.

## 3 `music21`

Given the rich history of music programming languages, and the demand for a simple-to-use, object-oriented software package to address musicological problems, `music21` was born. The software package is built in the Python programming language, which is lightweight, extensible, and modular (Cuthbert and Ariza 2010). As Cuthbert stated (2010):

> . . . [A]fter adding the system with from `music21` import *, straightforward tasks such as displaying or playing a short melody, getting a twelve-tone matrix, or converting from Humdrums Kern format to MusicXML can each be accomplished with a single line of code.

Not only are the Kern and the MusicXML formats both supported, but also the MuseData format as described by Walter B. Hewlett (1997). Furthermore, visualizers such as LilyPond and Finale Notepad are easily initiated with a simple *.show()* command from a parsed *corpus* object.

The installation of `music21` conveniently comes with a built in corpus for immediate analysis of digital scores. A selection of the artists available upon installation range from Johann Sebastian Bach, Ludwig van Beethoven, George Frideric Handel, Claudio Monteverdi, Franz Schubert, Arnold Schoenberg, and even John Coltrane (Cuthbert and Ariza 2010).

## 4 Uses

Because it was recently created, many of the published applications of the `music21` library have been produced by its creators. For example, Ariza and Cuthbert (2010) used the `music21` object-oriented framework to hierarchically model time signatures, beats, and meter. They further exploited the simple modular design of `music21` to explore the analytical and compositional of a network-based scale model (Ariza and Cuthbert 2011a). Lastly, they explored the idea of the musical *stream*—inherent to the `music21` object structure—for representing, filtering and transforming symbolic musical structures (Ariza and Cuthbert 2011b).

However, a couple of unpublished works in progress are worth mentioning. At McGill University, researchers are leveraging `music21` to search for Vertical Interval Successions over large sets of early classical scores (Antilla 2012). Similarly, in the Research Centre for Harpsichord Performance (CRIC), students have built a framework for implementing the accompaniment rules for the harpsichord based on the rules of the 18th-century music theorist Monsieur Saint de Lambert (Robertson 2012).

## 5 Example

In order to comprehend the simplicity of `music21`, a custom example is useful. The following script will load a random song from the repertoire of three very different composers that are included with the default `music21` installation. The script will parse the score of the random song selected, and then compute the pitch-class distribution for each of the songs:

```
from \texttt{music21} import *
from random import *
folkPaths = corpus.getComposer('essenFolksong')
coltranePaths = corpus.getComposer('coltrane')
bachPaths = corpus.getComposer('bach')
randInt = randint(0, len(folkPaths) - 1)
print "randInt is " + str(randInt)
folkSong = corpus.parse(folkPaths[randInt])
print "done parsing folk"
print "coltrane len is " + str(len(coltranePaths))
coltraneSong = corpus.parse('coltrane/giantSteps')
print "done parsing coltrane"
randInt = randint(0, len(bachPaths) - 1)
print "randInt is " + str(randInt)
bachSong = corpus.parse(bachPaths[randInt])
print "done parsing all"

#show songs
bachSong.show('text')
coltraneSong.show('text')
folkSong.show('text')
bachSong.show()
coltraneSong.show()
folkSong.show()

print "folk PitchClassDistribution:"
folkFE = features.jSymbolic.PitchClassDistributionFeature(folkSong)
folkF = folkFE.extract()
print folkF.vector

print "coltrane PitchClassDistribution:"
coltraneFE = features.jSymbolic.PitchClassDistributionFeature(coltraneSong)
coltraneF = coltraneFE.extract()
```

```
print coltraneF.vector

print "bach PitchClassDistribution:"
bachFE = features.jSymbolic.PitchClassDistributionFeature(bachSong)
bachF = bachFE.extract()
print bachF.vector
```

## 6 Conclusion

`music21`'s strength lies in its simplicity and its object-oriented design. With only a small amount of code, one can easily access a wealth of symbolic analysis tools. Its ability to parse and manipulate a wide array of formats, as well as to rapidly visualize digital scores give it the position as a potential leader in the future of programmatic analysis of music.

## References

Antilla, C. 2012. ELVIS source code, web resource. “https://github.com/crantila/vis”. Accessed 15 March 2013.

Ariza, C., and M. S. Cuthbert. 2010. Modeling beats, accents, beams, and time signatures hierarchically with music21 meter objects. In *Proceedings of the International Computer Music Conference*, 216–23.

Ariza, C., and M. S. Cuthbert. 2011a. Analytical and compositional applications of a network-based scale model in music21. In *Proceedings of the International Computer Music Conference*, 701–8.

Ariza, C., and M. S. Cuthbert. 2011b. The music21 stream: A new object model for representing, filtering, and transforming symbolic musical structures. In *Proceedings of the International Computer Music Conference*, 61–8.

Cuthbert, M. S., and C. Ariza. 2010. A multimodal approach to music transcription. In *Proceedings of the International Society for Music Information Retrieval*, 637–42.

Geiger, G. 2005. Abstraction in computer music software systems. Master's thesis, University of Pompeu Fabra.

Hewlett, W. B. 1997. Musedata: Multipurpose representation. In E. Selfridge-Field (Ed.), *Beyond MIDI: the Handbook of Musical Codes*, 402–7. Cambridge: MIT Press.

Huron, D. 1997. Humdrum and kern: selective feature encoding. In E. Selfridge-Field (Ed.), *Beyond MIDI: the Handbook of Musical Codes*, 375–401. Cambridge: MIT Press.

Robertson, H. 2012. Figure Divinator documentation, web resource. “http://www.music.mcgill.ca/˜hannah/CRIC/”. Accessed 15 March 2013.