

Abjad, A Python API for Formalized Score Control

Trevor Bača

Harvard

trevorbaca@gmail.com

Josiah Wolf Oberholtzer

Harvard

josiah.oberholtzer@gmail.com

Jeffrey Treviño

Carleton College

jeffrey.trevino@gmail.com

Victor Adán

Bank of America

vctradn@gmail.com

ABSTRACT

Place your abstract at the top left column on the first page. Please write about 150-200 words that specifically highlight the purpose of your work, its context, and provide a brief synopsis of your results. Avoid equations in this part.

1. INTRODUCTION

2. BACKGROUND & HISTORY

3. NOTATIONAL ISOMORPHISM

Abjad models objects on the page according to common practice notation.

Our musical model attempts to be isomorphic with notation, with one class per musical construct: Note, Rest, Chord, NoteHead, Articulation, TimeSignature, Slur and so forth.

[smalltalk?]

3.1 Notational aggregation

We assume notational primitives are the elements of composition. The act of composition then revolves around the iterative aggregation of notational primitives into arbitrarily complex score objects.

Abjad allows for the iterative aggregation of notational elements via Python's *mutable sequence* protocol. Score components can be appended, extended or inserted into other container-like score components.

```
>>> score = Score()
>>> staff_group = StaffGroup()
>>> upper_staff = Staff()
>>> lower_staff = Staff()
>>> score.append(staff_group)
>>> staff_group.extend([upper_staff, lower_staff])
>>> upper_staff.extend("c'4 d'4 e'4 f'4")
>>> lower_staff.extend([Chord("<a ef'>4."), Note("g8")])
>>> lower_staff.insert(0, Rest((1, 2)))
>>> show(score)
```

Copyright: ©2013 Trevor Bača et al. This is an open-access article distributed under the terms of the [Creative Commons Attribution 3.0 Unported License](#), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.



Note that Abjad makes visualizing notational artifacts simple. Any notational element or aggregate can be displayed at any time as a PDF via calls to its top-level `show()` function in near-publication quality.

Spanners such as slurs, beams and glissandi and indicators such as articulations and textual directions can be attached to score components via the `attach()` function.

```
>>> attach(Slur(), upper_staff[:])
>>> attach(Glissando(), lower_staff[1:])
>>> attach(Articulation('accent'), lower_staff[-1])
>>> show(score)
```



Abjad attempts to be compositionally agnostic. By providing simple and unambiguous means of gradually aggregating arbitrarily complex score objects, we encourage users to develop their own personal approach.

3.2 Explicit notational modeling

Abjad models notation explicitly. All notational primitives expressed by Abjad must conform to the principles of common practice notation. When compositional inputs cannot be expressed in terms of these principles, we provide affordances for massaging them into valid notational states.

For example, Abjad expresses the durations of all score components in terms of rational values – fractions and integers – rather than floating point numbers. Likewise Abjad expresses all pitches in terms of triples of diatonic note names, accidentals and octave numbers, rather than MIDI numbers or frequencies. While we provide alternative representations of pitch and rhythm, as well as affordances for moving between them, the format actually stored in and used by score components for rendering notation is always the most notationally-explicit.

3.3 Written, assignable and prolated durations

All Note, Chord and Rest objects in Abjad must be instantiated with a duration corresponding to the written glyphs on the page – a *written* duration.

Written durations must be *assignable*, a category we invented to model durational initialization. Durational assignability describes whether a duration can be represented as a power-of-two flag count combined with zero or more dots. $1/4$, $3/16$ and $7/16$ are assignable durations while $5/32$, $9/8$ and $1/12$ are not.

Non-assignable durations cannot be represented in common practice notation by a single glyph. They require two or more glyphs with assignable durations tied together, for the score component to be tupletted, or both.

Abjad will not automatically render a single note with a duration of $5/16$ as two or more notes tied together. We consider such behavior to be too implicit. There are too many potentially compositionally valid ways to render a duration such as $5/16$ into a series of tied assignable durations: $1/4 + 1/16$, $3/16 + 2/16$, $2/16 + 3/16$, $1/16 + 1/4$, $1/8 + 1/8 + 1/16$ etc. Instead we provide affordances for generating tied notes from non-assignable durations. One such affordance is our `scoretools.make_notes()` function, which chooses smart defaults for generating tied glyphs from otherwise un-notateable input.

```
>>> selection = scoretools.make_notes("c'", [(5, 16)])
>>> staff = Staff(selection)
>>> show(staff)
```



All score components also have a *prolated* duration - the product of their written duration and their *prolation*. Prolation is the cumulative product of all the *multiplier* of every tuplet found in the *parentage* of a score component. A score component's prolation depends on its location in the score hierarchy, and is not an inherent property of itself independent that hierarchy.

Three Note objects each having a prolated duration of $1/12$ can be represented as either three $1/16$ notes in a 3:4 tuplet or as three $1/8$ notes in a 3:2 tuplet. As all Abjad Note objects must have an assignable written duration, the three notes above must have written durations of either $1/8$ or $1/16$, and the tuplet must be correspondingly an explicit diminution or augmentation to provide the desired prolation of $2/3$ or $4/3$.

```
>>> selection = scoretools.make_notes("c'", [(1, 12)] * 3)
>>> tuplet = selection[0]
>>> show(tuplet)
```



```
>>> tuplet.toggle_prolation()
>>> show(tuplet)
```



The durational information of any aggregate score object in Abjad is therefore always explicit and unambiguous with regard to its notational reality.

4. RELATIONSHIP MODELING

5. SCORE ADDRESSABILITY

6. EXTENSIBILITY

7. EMBEDDABILITY

Abjad is an importable Python library. It can be used in whole or in part as a component of any Python-compatible system. Abjad has few Python package dependencies and is not bound to any specific user application or graphic user interface. These qualities make Abjad an ideal project ideal for embedding in other software systems.

For example, Abjad supports IPython Notebook¹, a web-based interactive computational environment combining code execution, text, mathematics, plots and rich media into a single document. Notational output from Abjad can be transparently captured and embedded directly into an IPython Notebook which has loaded Abjad's IPython Notebook extension. Calls to Abjad's `show()` and `play()` are intercepted, and the rendered graphic and audio outputs are embedded directly into the Notebook along with the generating code. This allows scholars to quickly and intuitively create music texts which can be shared, edited and executed by other IPython users.

8. OPEN SOURCE

Acknowledgments

You may acknowledge people, projects, funding agencies, etc. which can be included after the second-level heading "Acknowledgments" (with no numbering).

¹ <http://ipython.org/notebook.html>