NIME 2025 – Canberra, Australia

SMucK: Symbolic Music in ChucK

Alex Han, Kiran Bhat, Ge Wang

Center for Computer Research in Music & Acoustics (CCRMA) - Stanford University

What is SMucK?

SMucK is a library and workflow for creating music with symbolic data in the ChucK programming language. It extends ChucK by adding the following core features:

1) *SMucKish:* Text-based notational language for score input
2) *ezScore*: Data structures to represent score information
3) *ezInstrument*: Custom classes for linking sound design to scores
4) *ezScorePlayer*: Score playback and flexible time handling

Why SMucK?

SMucK makes programming in ChucK more musical.

Before SMucK, there were no built-in tools and abstractions in ChucK for many common musical concepts:

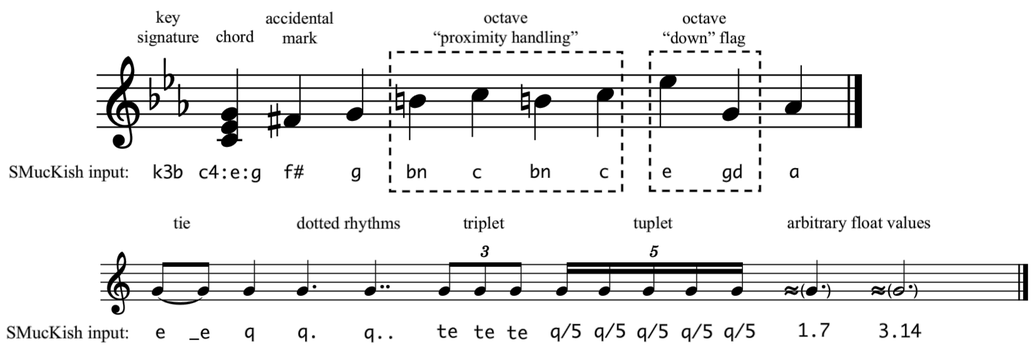
pitch polyphony tempo articulation rhythm measures scales dynamics notes chords form

With SMucK, it is much easier to compose and program music using these concepts.

SMucK goes beyond static representation of score material, taking advantage of ChucK's strongly-timed, concurrent programming model to allow for dynamic and precise control over playback. SMucK scores are editable on-the-fly, and multiple scores can be played back concurrently, with independent timing control. These features make SMucK well suited for exploring compositional ideas, live-coding, and designing interactive systems.

SMucKish Input Syntax

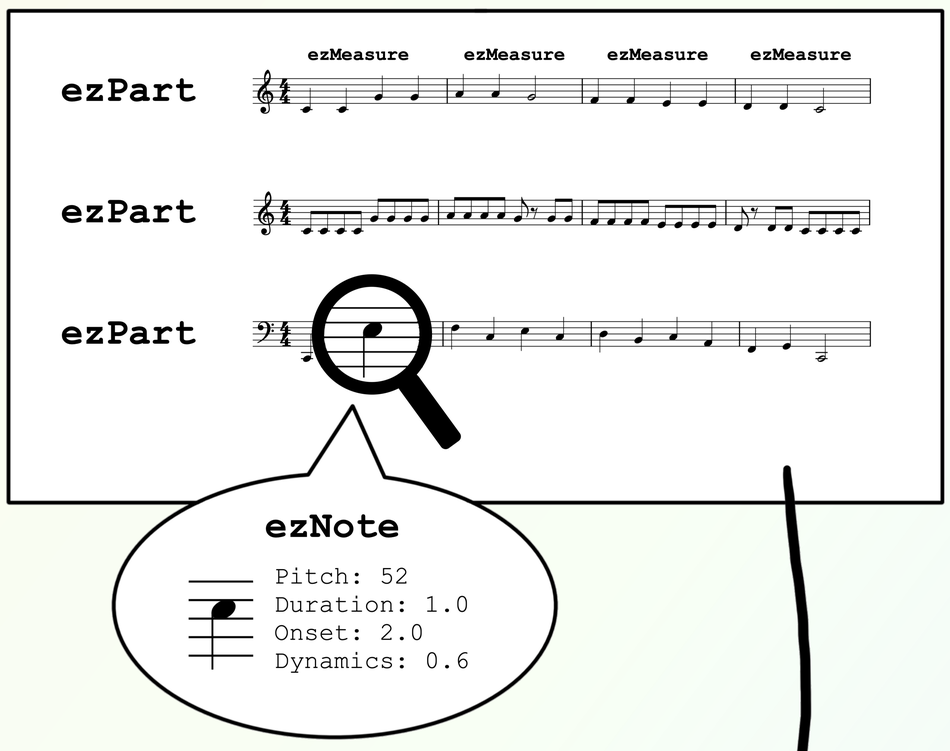
SMucKish takes inspiration from other notation systems like Lilypond, ABC, and especially Leland Smith's SCORE. It is designed to be compact, readable, and based on familiar notation conventions. Here are some examples of pitch and rhythm specifications in SMucKish:



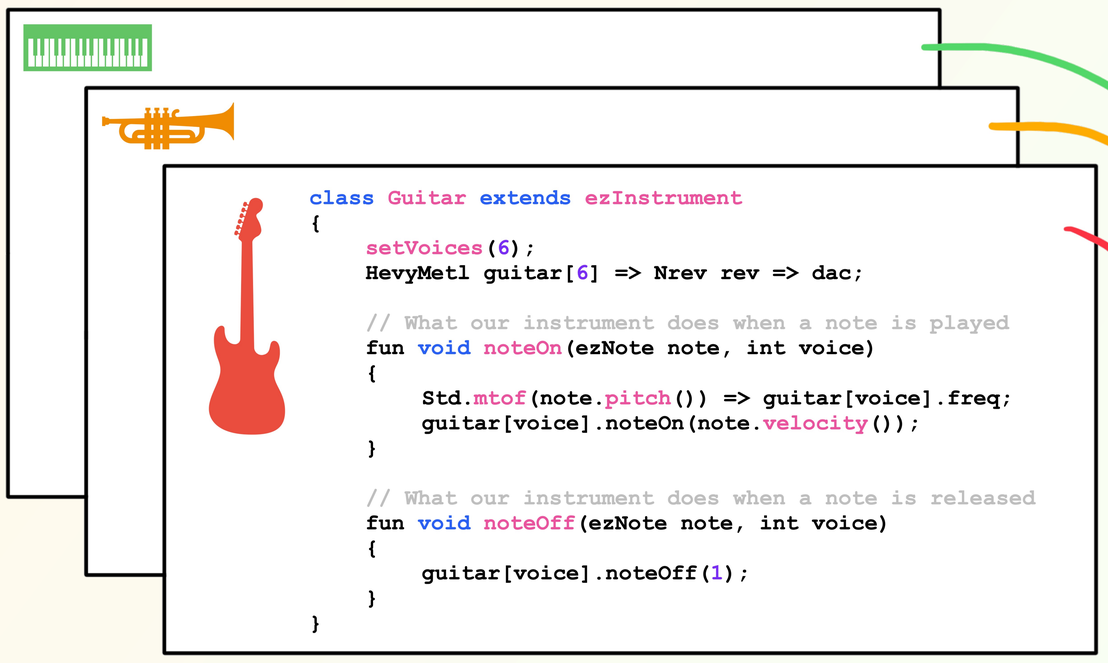
SMucKish also allows users to write dynamics, text annotations, or arbitrary expression values as additional layers. Users can also enter names of chords (e.g. "Cmaj7", "Bb7#9b13") and scales (e.g. "minor", "mixolydian").

SMucKish input can be parsed separately into float arrays or all at once into *ezNote* objects, which in turn can populate *ezMeasures*, *ezParts*, and *ezScores*

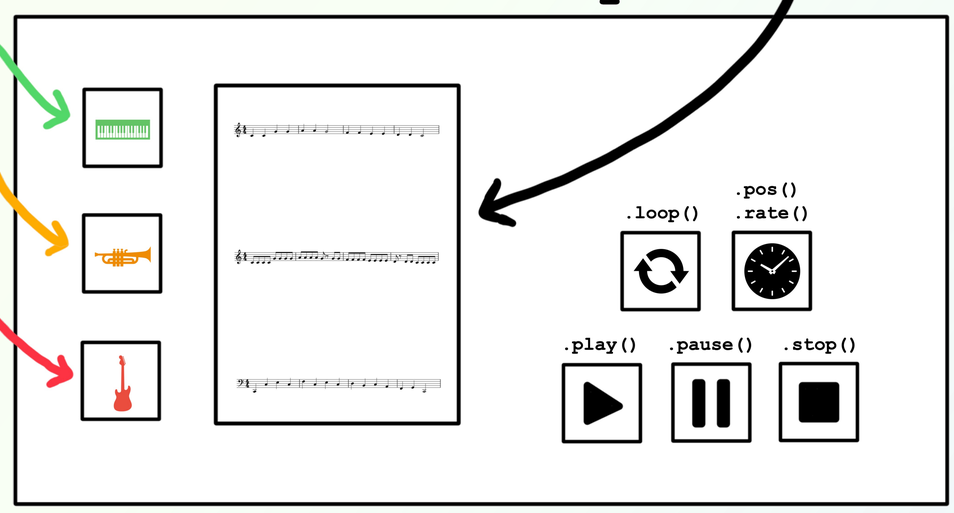
ezScore



ezInstruments



ezScorePlayer



Workflow

Here is an example of the SMucK workflow from composition to playback:

1) Write an ezScore with SMucKish or import from MIDI/musicXML
2) Create ezInstruments specifying sound synthesis chains
3) Instantiate an ezScorePlayer, enabling playback of the ezScore using their ezInstruments
4) Play the score!

```
@import "smuck"
1 ezScore score("a b c d");
2 myInstrument inst => dac;
3 ezScorePlayer player(score);
  player.setInstrument(0, inst);
4 player.play();
  score.duration() => now;
```

Conclusions

The goal of SMucK is to draw upon the affordances of many existing symbolic music information systems: the compactness and readability of notation-focused languages, the interactivity with hierarchically-organized score data found in graphic score editors and DAWs, and the customizability and dynamic computing of live-coding frameworks. While SMucK is not the first system to do any one of these things, it offers an accessible, versatile, extensible, unified workflow for working with symbolic music within ChucK's unique programming model.

Our project is still in its infancy. It has not been widely tested and used by musicians and coders, and its features are still actively being improved upon. Since SMucK's official release earlier this year, students at CCRMA have started using SMucK in their creative projects. Students' feedback on the system on both the technical and artistic levels will inform our ongoing development directions. We are currently working on adding more tools for dynamically manipulating score contents, allowing more expressive information to be encoded, and supporting integration with other software, languages, and plugins.

Try SMucK!

SMucK
Symbolic Music in ChucK.

CCRMA

Stanford University

NIME 2025