.

CsoundAC User Guide (Mostly for Python)

Michael Gogins
michael.gogins@gmail.com

February 11, 2025

1 Introduction

CsoundAC is a class library and software toolkit designed to support the algorithmic composition of music. CsoundAC has C++, Python, and JavaScript application programming interfaces (APIs). This ***Guide*** is based on the Python interface.

References herein are all in the form of colored hyperlinks to resources on the World Wide Web. Reference documentation for CsoundAC is here.

The major features of CsoundAC are:

- Music graphs, a hierarchical representation of musical scores and processes, modeled upon the computer graphics idea of scene graphs.
- A variety of nodes for use in music graphs, including random variables, dynamical systems, Lindenmayer systems, iterated function systems, and others.
- Operations on scores using aspects of mathematical music theory, including chord spaces, chord transformations, voice-leading, and automatic modulations.
- As the name suggests, CsoundAC interfaces closely with Csound. However, CsoundAC can also be used with any other computer music system that supports Python scripting, including digital audio workstations (DAWs) such as Reaper.

The core of CsoundAC is written in platform-neutral standard C++. The CsoundAC repository also includes a Python interface to CsoundAC. A JavaScript interface to a WebAssembly (WASM) build of CsoundAC (and of Csound) is available in the csound-wasm repository.

2 Getting Started

2.1 Installation

Prebuilt binaries of CsoundAC for macOS and Linux are available as GitHub releases. These are built for a specific version of Python, and will work only with that version. If you don't have the correct version of Python, you should install that version alongside your existing Python.

Linux binaries can also be built by the user as follows:

1. Clone the csound-ac repository, and open a terminal in the repository's root directory.
2. Install dependencies by running the `update-dependencies.sh` script.
3. Perform a fresh build by running the `clean-build-linux.sh` script.

2.2 Configuration

2.2.1 Configuring CsoundAC

Copy `_CsoundAC.so` and `Csound.py`, as well as `GeneralizedContextualGroup.py`, to the `site-packages` directory for the version of Python with which CsoundAC was built,, e.g. on macOS it could be `/opt/homebrew/lib/python3.12/site-packages/_CsoundAC.so`.

Test your installation by running the correct version of Python and importing the CsoundAC and GeneralizedContextualGroup modules. There should be no error.

```
michaelgogins@macbookpro ~/csound-ac % python3.12
Python 3.12.8 (main, Dec  3 2024, 18:42:41) [Clang 16.0.0 (clang-1600.0.26.4)] on
    darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import CsoundAC
>>> import GeneralizedContextualGroup
>>>
```

2.2.2 Configuring ctcsound

Each published release of Csound comes with a Python interface to the complete Csound API, defined in the `ctcsound.py` module. This module will work with any version of Python, but is built for a specific version of Csound. To enable ctcsound:

1. Go the Csound GitHub repository, select the `csound6` branch, and copy or download the `interfaces/ctcsound.py` file to the `site-packages` directory for the version of Python that you will use, e.g. `/opt/homebrew/lib/python3.12/site-packages/ctcsound.py`.
2. Test your installation by running the correct version of Python and importing the ctcsound module. There should be no error.

```
michaelgogins@macbookpro ~/csound-ac/user-guide % python3.12
Python 3.12.8 (main, Dec  3 2024, 18:42:41) [Clang 16.0.0 (clang
    -1600.0.26.4)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from ctcsound import *
>>>
```

2.2.3 Configuring Reaper

Additional configuration is required to enable Python and CsoundAC in Reaper. To enable Python scripting:

1. In Reaper, open the *REAPER* menu *Settings...* dialog (might be *Configuration...* on other versions of Reaper).
2. Go to the end of the list of settings, and select *ReaScript*.
3. In the ReaScript settings, check the *Enable Python for use with ReaScript* box.
4. On macOS:
   a) In the *Force ReaScript to use specific Python .dylib* text field, enter the filename of the Python shared library for the version required by CsoundAC, e.g. `libpython3.12.dylib`.
   b) In the *Custom path to Python dll directory* field, enter the full path to the Python library root directory, e.g. `/opt/homebrew/Cellar/python@3.12/3.12.8/Frameworks/Python.framework/Versions/3.12/lib`. You can use the *Browse* button to select this directory.
   c) If the configuration is correct, you will see something like *Python: libpython3.12.dylib installed*.
5. On Linux:
   a) In the *Force ReaScript to use specific Python .so* text field, enter the filename of the Python shared library for the version required by CsoundAC, e.g. `libpython3.12.so`.
   b) In the *Custom path to Python dll directory* field, enter the full path to the Python library root directory, e.g. `/usr/lib/libpython3.12.so` or `/usr/lib/x86_64-linux-gnu/libpython2.12.so`. You can use the *Browse* button to select this directory.
   c) If the configuration is correct, you will see something like *Python: libpython3.12.so installed*.

To enable CsoundAC:

1. Exit Reaper, and run it again.
2. Use the *Actions* menu, *New action...* dialog, "Load ReaScript..." dialog to load the `ac_reaper.py` script.
3. Create a track with a blank MIDI item, and select that item.
4. Use the*Actions* menu, *Show action list...* dialog, select the *ac_reaper.py* script, and click on the *Run* button.
5. You should see the *ReaScript console output* dialog, with output from the *ac_reaper.py* script. It should look something like this:

```
Hello, Reaper!
Selected item: <class 'str'> (MediaItem*)0x000000014383A400
Using active take of selected MIDI item.
Start time of the selected MIDI item: 0.00 seconds.
Event count: note_count: 4 cc_count: 0 text_sysex_count: 0
Cleared all contents from the MIDI item.
Inserting note: Start: 0 Start PPQ: 0.0 End PPQ; 960.0 Key: 60 Velocity: 100
Inserting note: Start: 0.5 Start PPQ: 960.0 End PPQ; 1920.0 Key: 62 Velocity
    : 100
```

```
Inserting note: Start: 1 Start PPQ: 1920.0 End PPQ; 2880.0 Key: 64 Velocity:
    100
Inserting note: Start: 1.5 Start PPQ: 2880.0 End PPQ; 5760.0 Key: 67
   Velocity: 100
MIDI notes added successfully! Check the editor.
```

6. If you see the generated notes in the MIDI editor, and there is no error message, then your configuration is correct.
7. If you see the generated notes but also see *WARNING: Could not import CsoundAC!*, then Reaper can run Python code, but the installation or configuration for CsoundAC is not correct. The most likely causes are that the custom path to the Python library is not correct, or CsoundAC files are not located in the correct `site-packages` directory.

3 Examples

The following examples proceed step by step, repeating each example in standalone Python, Csound's score bin facility, and Python in Reaper:

1. Generate a chromatic scale using just Csound and Python.
2. Generate the same chromatic scale using CsoundAC.
3. CsoundAC compositions demonstrating some capabilities of CsoundAC.

For insight into the code, go to the hyperlinked source code or open the examples in an editor, and read them in order; there are many explanatory comments.

3.1 Basic Python Usage

Here, the same very basic example is run in three environments. All three examples use the same Csound orchestra, `basic.csd`, which is defined in the Python code as a Python string literal (`(r"some␣string␣with␣perhaps␣many␣lines␣and␣escape␣characters"`). All three examples use a simple `for` loop to generate an ascending chromatic scale of notes. The only differences between the examples are in how they are run, and how they interface with Csound. CsoundAC is not used — these examples show only the most basic, low-level method of generating a score.

3.1.1 Running in Python: `basic-python.py`

Running in Python has the advantage of being easy to understand. A composition is just a Python program. This example uses the ctcsound module to perform the example.

Open a terminal, change to the `csound-ac/user-guide` directory, and execute `python3.12 basic-python.py`

3.1.2 Running in Csound: `basic-score-bin.csd`

Running in Csound has the advantage of requiring no additional installation or configuration apart from Csound, Python, and CsoundAC. All of the elements of a composition are defined in one .csd file. In this case the Csound orchestra is defined outside of the Python code, which is embedded in the .csd file.

Thus, code for running Csound is not needed, as Csound is already running; instead, the script writes the generated score to a temporary filename, and Csound reads that score and performs it.

Open a terminal, change to the `csound-ac/user-guide` directory, and execute `csound basic-score-bin.csd`.

3.1.3 Running in Reaper: `basic-reascript.py`

Running in Reaper has the advantages of automatically providing a piano roll score, or even music notation, for generated scores; enabling other synthesis plugins besides Csound to render CsoundAC scores; and in general, integrating with contemporary music production practice. The Csound orchestra is defined in the state of a CsoundVST3 plugin, which receives MIDI input from the DAW. The compositional code is defined in a Python ReaScript file, which is loaded into Reaper as an Action.

This separation of concerns between composition and synthesis enables CsoundAC to send MIDI to the DAW and to other plugins, and for Csound to receive MIDI and audio from any track or plugin in Reaper.

Use the *Actions* menu, *New action...* dialog, "Load ReaScript..." dialog to load the `basic-reascript.py` script. Create a track with a blank MIDI item, and select that item. Use the *Actions* menu, *Show action list...* dialog, select the *basic-reascript.py* script, and click on the *Run* button.

You should see the *ReaScript console output* dialog, with output from the script. You should also see the notes of the chromatic ascending scale in the MidiItem. You can then assign a synthesizer, which could be CsoundVST3 or another synthesizer, to play the notes.

3.2 Minimal CsoundAC Examples

These three examples re-create the three basic examples above, changing them only to generate the ascending chromatic scale as a CsoundAC ScoreNode object, and to render that using CsoundAC's integration with Csound.

1. Running in Python: `csoundac-basic-python.py`.
2. Running in Csound: `csoundac-basic-score-bin.csd`.
3. Running in Reaper: `csoundac-basic-reascript.py`.

3.3 Compositional Examples

> In the following, specifically compositional remarks are framed like this.

3.3.1 How CsoundAC Works

CsoundAC is based upon the concept of a *music graph,* which is a tree (directed acyclical graph) of Nodes in score space. This concept is quite similar to the concept of a *scene graph* in 3-dimensional computer graphics, in which objects in a scene are similarly organized as a tree of nodes in 3-dimensonal space. Score space, however, has 12 dimensions:

TIME Time in seconds relative to the origin of the space.

DURATION The duration of this event in seconds.

STATUS MIDI status number, normally 144 for "note on."

INSTRUMENT MIDI channel number or Csound instrument number - 1.

KEY MIDI key number from 0 to 127 in semitones; may have a fractional part to represent microtones; middle C is 60.

VELOCITY MID velocity number from 0 to 127; can loosely be considered decibels of sound pressure level.

PHASE Phase of the sound, used mostly for Events that represent grains of sound.

PAN Location in physical space from left to right, in the interval [0, 1] where .5 is the center.

DEPTH Location in physical space from in front to behind, in the interval [0, 1] where .5 is the center.

HEIGHT Location in physical space from below to above, in the interval [0, 1] where .5 is the center.

PITCHES Represents a pitch-class set as a sum of powers of 2 where each power represents one of the pitch-classes in 12 tone equal temperament.

HOMOGENEITY Set to 1 to enable transformation of Events using homogeneous matrix transformations.

A point in score space most often represents a musical note, but can also represent a chord or scale, a change in a control parameter, or even a grain of sound. The semantics of these dimensions is based on MIDI channel messages, but has floating-point precision and adds duration and other fields.

The following algorithm is used by CsoundAC to generate scores. A composition is an instance of the MusicModel class, which contains a tree of child Nodes representing the music graph, a Csound orchestra, and an embedded instance of Csound. The algorithm performs a depth-first traversal of the music graph by recursively calling each child's `void Node::traverse(const Eigen::MatrixXd &global_coordinates, Score &global_score)` method:

- Multiply the global coordinates by the local coordinates to obtain composite coordinates.
- Descend into each of this Node's child Nodes with the composite coordinates and an empty Score.
- Call `Node::transform` to transform any or all Events produced by the child nodes, which are in the composite coordinate system.
- Add the child Events to the global score.
- Create another empty Score in which to optionally generate new Events, and call `Score::transform` to move these new Events into the composite coordinate system.
- Add these generated Events to the global score.

In standalone Python, once the global Score has been generated, the MusicModel uses an embedded instance of Csound to compile its orchestra, translates the global CsoundAC Score to a Csound score, and renders that score with the orchestra.

In Csound using the `<CsScore bin=python3>` facility, the generated Score must be translated to a Csound score and written to the temporary filename.

In Python ReaScript, the generated Score must be passed to the `ac_reaper.score_to_midiitem` function.

The following three examples demonstrate some typical uses of CsoundAC. Two implementations are provided for each example: as standalone Python, and as Python ReaScript for Reaper.

3.3.2 Transforming Source Material: `python-trans-scale.py`

This piece takes the ascending chromatic scale of the previous examples, and uses CsoundAC to transform and elaborate it.

1. Play the scale.
2. Repeat that scale, shuffled so that notes are in random order.
3. Repeat that transformed scale, and randomize the note durations.
4. Repeat that transformed scale, quantizing the note onsets and durations.
5. Repeat that transformed scale three times, as a canon at the third, after first cutting down the jumps in the transformed scale in order to make the canon easier to hear.
6. Repeat that canon, and apply to it semi-randomized chord progressions and modulations.

Read the code for more details.

3.3.3 Translating Images to Scores: `python-imagery.py`

This piece takes a high-resolution digital photograph, and uses CsoundAC to:

1. Load the photograph in an ImageToScore node.
2. Set a threshold to filter out much of the image, and use only the brighter features.
3. Set a maximum number of simultaneous voices.
4. Call `ImageToScore.generateLocally` to translate the brighter features to CsoundAC Events (notes). Hue (color) is instrument, and Value (brightness) is loudness. Saturation is not used.
5. Add the ImageToScore node to a Rescale node.
6. Set appropriate ranges for the various dimensions of the score space in the Rescale node. *Please note: time is not rescaled!* That is so that the duration of the score created by the ImageToScore node can be used to determine the intervals between chords.
7. Use the CsoundAC Scale class to generate a series of chord progressions and modulations.
8. Add the resulting Chords to a VoiceleadingNode, which will apply them to the Events in the generated and rescaled score.

9. Add the Rescale node to the VoiceleadingNode.
10. Add the VoiceleadingNode to the MusicModel.
11. Call `MusicModel.generate` to the generate the score, applying each node to the notes produced by its children.
12. Set the final generated score to the desired length.
13. Call `MusicModel.perform` to render the score using the Csound orchestra.

3.3.4 Lindenmayer Systems in Chord Space: `python-duo-dualities.py`

This piece uses a Lindenmayer system defined in the voice-leading spaces of Callendar, Quinn, and Tymoczko to implement the Generalized Contextual Group of chord transformations identified by Fiore and Satyendra. Here I use the Generalized Contextual Group to capture the duality of major and minor so constitutive of Western music, in an abstract and generative form that is not specifically "tonal." This Lindenmayer system, presented by me at the 2009 International Computer Music Conference, generates a score as movements of chords by transformation through chord space and time, writing notes of the chords to create the score.

The piece is a modified and elaborated form of *Two Dualities* by me, presented at the 2010 International Computer Music Conference in the listening rooms as part of 360 degrees of 60x60 (Crimson Mix).

For some hints as to the Lindenmayer system commands in `GeneralizedContextualGroup.py`, see the comments in the source code.