User Guide
Csound 6.11.0 for Android

Michael Gogins

6 December 2018

Introduction

This is an extremely basic guide to using the Csound for Android app. It should be just enough to get you started if you already have some experience with Csound or, at least, some other “Music N” type software synthesizer.

The Csound for Android app is an Android version of Csound, an audio and music synthesis and processing language of great power, created by Barry Vercoe in 1984, maintained with complete backward compatibility since then, and widely used and taught for the creation of electroacoustic and computer music. Desite its age, Csound continues to be actively developed. It has many advanced features for sound synthesis including user-designable instruments, an easy to learn programming language with a type system, several phase vocoders, several sample players, a plethora of digital oscillators, filters, and envelope generators, transparent multi-threading for high performance on multi-core systems, and an embedded Lua just-in-time compiler.

The Csound for Android app is by Victor Lazzarini, Steven Yi, Michael Gogins, and others. The app contains virtually all of Csound, including some of the more frequently used plugin opcodes:

- The signal flow graph opcodes, for chaining instruments into effects chains and so on.
- The Fluidsynth opcodes for playing SF2 samples.
- The libstdutil library.
- The Lua opcodes, which embed LuaJIT and enable arbitrary Lua code to execute in Csound either for signal processing or for score generation or, indeed, for any purpose. Particularly amazing is that LuaJIT has a foreign function interface that is capable of loading any shared library that Android permits, and calling any exported function in that library.
- The scanned synthesis opcodes.
- The Open Sound Control (OSC) opcodes which allow Csound to communicate over networks with other software in a low-latency and flexible manner.
- The Doppler opcode.

User Interface

The menu buttons perform the following functions:

- ***New*** – Creates a blank template CSD file in the root directory of the user's storage for the user to edit. The CSD file will be remembered and performed by Csound.
- ***Save as...*** – Saves the current CSD file under a new name, and opens the new file.
- ***Open*** – Opens an existing CSD file in the root directory of the user's storage. The user's storage filesystem can be navigated to find other files.
- ***Edit*** – Opens a text editor to edit the current CSD file. Be sure to save the file before you perform it! I recommend the free, open source Jota text editor on smartphones and, though I

haven't tried Jota on tablets, it probably works well there as well.

- ***Start/Stop*** – If a CSD file has been loaded, starts running Csound; if Csound is running, stops it. If the `<CsOptions>` element of the CSD file contains `-odac`, Csound's audio output will go to the device audio output. If the element contains `-osoundfilename`, Csound's audio output will go to the file `soundfilename`, which should be a valid Linux pathname in the user's storage filesystem.

There are actually two types of widget area: one with pre-defined widgets, and one with user-defined widgets.

Menu

The main menu (top right-hand button on the title bar) includes the following choices:

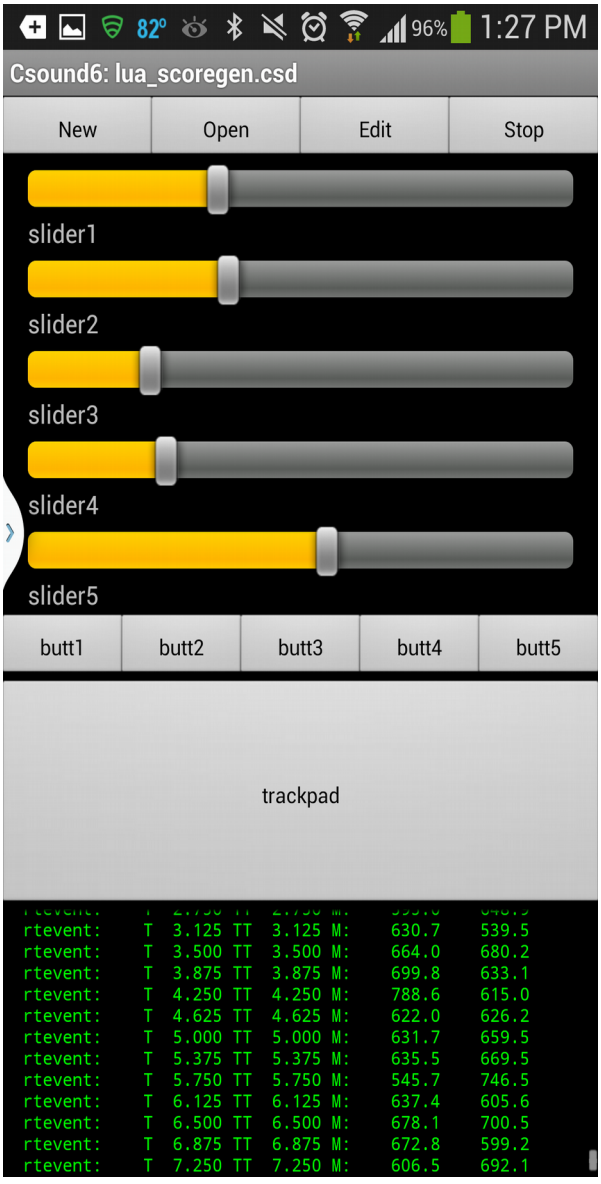
- ***Examples*** – Opens a submenu with various built-in example pieces. These are saved in your device's external public storage Music directory, and then run.
- ***User guide*** – Opens this basic user guide.
- ***Csound help*** – Opens a browser to show the complete Csound Reference Manual online.
- The ***Settings*** menu includes the following choices. The values of the settings are saved on the device, and restored at the beginning of the next session.
    - ***Plugins directory*** – Sets the value of the OPCODE6DIR64 environment variable.
    - ***Output directory*** – Sets the value of the SFDIR environment variable.
    - ***Samples directory*** – Sets the value of the SSDIR environment variable.
    - ***Analysis directory*** – Sets the value of the SADIR environment variable.
    - ***Include directory*** – Sets the value of the INCDIR environment variable.
    - ***Screen Layout*** – Configures the high-level layout of the Csound for Android screen, as follows:
        - ***Message console*** – Displays only a scrolling console of Csound's diagnostic messages.
        - ***HTML*** – Displays only HTML in Csound's embedded Web browser (see ***User-Defined Widgets***, below).
        - ***HTML with message console*** – Displays HTML at the top and a small message console at the bottom.
        - ***Widgets*** – Displays only the predefined widgets (see below).
        - ***Widgets with message console*** – Displays predefined widgets and the top and a message console below.
- ***About Csound*** – Opens a browser to show the official Csound Web page online.

Predefined Widgets

The widgets are assigned control channel names `slider1` through `slider5`, `butt1` through `butt5`, `trackpad.x`, and `trackpad.y`. In addition, the accelerometer on the Android device is available as `accelerometerX`, `accelerometerY`, and `accelerometerZ`.

The values of these widgets are normalized between 0 and 1, and can be read into Csound during performance using the `chnget` opcode, like this:

```
kslider1_value chnget "slider1"
```



User-Defined Widgets

User-defined widgets are defined in an HTML file (Web page), or in a new `<html>` element of the CSD file. This can in principle contain any valid HTML5 code, including document elements, JavaScript, and styles. JavaScript event handlers can be attached to any elements, including range inputs (sliders). Such event handlers can call Java methods of tthe `CsoundObj` object (`csound`), for example to get or set control channel values, or to send new score events to Csound.

The ability to send score events to Csound from JavaScript means that JavaScript can be used as a high-level programming language in the CSD file for the purposes of algorithmic composition, even

interactive composition. All methods callable from `CsoundAppActivity` and `CsoundObj`are marked with the `@JavascriptInterface` attribute in the Csound for Android source code. For a basic example of a user-defined widget setup, see the `Drone-HTML5.csd` example file. Here is a snippet showing a slider's `onInput` event handler setting a Csound control channel:

```
<script>
function gk_Reverb_FeedbackUpdate(value) {
var numberValue = parseFloat(value)
csoundApp.setControlChannel('gk_Reverb_Feedback', numberValue);
}
</script>
```

Examples

The menu of the Csound app features a number of built-in examples. Selecting an example will cause its file or files to be copied to the app's external public storage Music directory and loaded into the app, ready to be performed or edited. Not all of the examples use the widgets. The examples demonstrate not only some techniques for using the Csound6 Android app, but also a few of the many different ways of making music with Csound.

When I first encountered the Csound app, I was very impressed. Now that I have been able to contribute to its development, I have come to realize that a high end smartphone, not to mention a tablet, is in every sense of the word a real computer. The limits to what can be programmed on it are indefinable. On a high-end personal computer, it is easier to type, and Csound runs quite a bit faster; but there is no *essential* difference between running Csound on a computer and running it on a smartphone.

A Tutorial Example

This example will take you through the process of creating a new Csound piece, step by step. Obviously, this piece is not going to reveal anything like the full power of Csound. It is only intended to get you to the point of being able to create, edit, and run a Csound piece that will actually make sound on your Android device – from scratch.

Before you get started, install the Jota text editor on your device. Other text editors might work with the Csound app, but this one is known to work.

Run the Csound6 app...

Press the **New** button. You should be presented with an input dialog asking you for a filename for your piece. Type in `toot.csd`, and press the **Ok** button. The file will be stored in the root directory of your user storage on your device. You can save the file to another place using Jota's **File** menu, if you like.

The text editor should open with a “template” CSD file. Your job is to fill out the minimum to hear something.

Create a blank line between `<CsOptions>` and `</CsOptions>`, and type `-odac -d -m3`. This means send audio to the real-time output (`-odac`), do not display any function tables (`-d`), and log some informative messages during Csound's performance (`-m3`).

Create a blank line between `<CsInstruments>` and `</CsInstruments>` and type the following text:

```
sr = 44100
ksmps = 10
```

```
nchnls = 1
instr 1
asignal oscil 10000, 440
out asignal
endin
```

This is just about the simplest possible Csound orchestra. The orchestra header specifies an audio signal sampling rate of 44,100 frames per second, with 10 audio frames per control signal sample, and one channel of audio output. The instrument is just a simple sine oscillator. It plays a loud tone at concert A.

Create a blank line between `<CsScore>` and `</CsScore>` and type:

```
i1 0 5
```

This means play instrument 1 starting at time 0 for 5 seconds.

Press the text editor's **Save** button and then its **Quit** button.

Press the Csound app's **Start** button. You should hear a loud sine tone for 5 seconds.

If you want to save your audio output to a soundfile named `test.wav`, change `-odac` above to `-o/sdcard/test.wav`.

That's it!