User Guide
Csound for Android

Michael Gogins

10 May 2019

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.

Changes in This Version

This version of Csound for Android features many changes and improvements:

- The app is now built for 64 bit CPU architecture.
- Audio input and output now are implemented using Google's Oboe library, which encapsulates differences between audio drivers on different Android platforms. Oboe is integrated with Csound in a new CsoundOboe C++ class.
- The app now has a built-in text editor based on CodeMirror. The editor includes find and replace functions.
- The user interface is tabbed for instantly switching between editing and performing.
- The ***Csound Reference Manual*** appears in its own tab.
- Embeddable Common Lisp is available, as well as the nudruz and Common Music libraries, to support algorithmic composition in Common Lisp.
- The software repository and build system have been simplified. Csound for Android and all native libraries and other dependencies are built directly from Android Studio.

Introduction

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, and transparent multi-threading for high performance on multi-core systems.

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 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.

In addition, Csound for Android now includes Embeddable Common Lisp along with the nudruz and Common Music libraries to support algorithmic composition in Lisp.

User Interface

The Csound for Android user interface uses tabs for easy and swift changes of context, for example between editing and performing. All tabs share the app's builtin instance of Csound. This enables, for example, a piece to be written in HTML5 but utilize the builtin widgets, execute Common Lisp code, and so forth. The tabs are:

- ***Editor*** – Provides a built-in text editor based on CodeMirror, with find and replace functionality.
- ***Messages*** – Shows all diagnostic and informative messages printed by Csound.
- ***HTML*** – Provides a built-in HTML5-enabled Web browser for the display of custom user interfaces, computer graphics and animations, and so on.
- ***Widgets*** – Provides a predefined set of 5 sliders, 5 buttons, and a trackpad that send updates to predefined Csound control channels. These widgets are backwards compatible with previous versions of the Csound for Android app.
- ***Help*** – Provides an embedded browser for the ***Csound Reference Manual***.
- ***About*** – Provides an embedded browser for the Csound home page at http://csound.com, enabling users to explore many uses and features of Csound.

The task menu of Csound for Android provides the following functions:

- ***New*** – Creates a blank template Csound .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.
- ***Open*** – Opens a file for editing. The file can be a Csound .csd file, an HTML5 .html file, or a Common Lisp .lisp file. Each of these languages has access to the same instance of Csound that is exposed in the HTML tab or to the widgets in the Widgets tab.
- ***Save as...*** – Saves the current .csd or other file being edited under a new name.
- ***Save*** – Saves the current .csd or other file being edited to the user's storage.
- ***Run/Stop*** – If Csound is not running, starts a performance; if Csound is running, stops the performance. If the file is an .html file, it is reloaded in the HTML tab which has access to a Csound object encapsulating the app's builtin instance of Csound. If the file is a .lisp file, it is

executed by an embedded Common Lisp runtime that has access to the app's instance of Csound as a native pointer. If the Csound options include `-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.

- ***Find...*** – Search for a string of text in the editor.
- ***Replace…*** – Replace a string of text in the editor.
- ***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.
- ***Settings*** – 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.

Predefined Widgets

The predefined widgets on the ***Widgets*** tab 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 the `CsoundOboe` 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 an .html file for the purposes of algorithmic composition, even interactive composition. All methods callable from `CsoundAppActivity` and `CsoundOboe` 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.

Run the Csound 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 , if you like.

The text editor should now contain 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.

Invoke the **Save** menu item.

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!

Audio Performance Issues

The Android operating system has only recently been given an audio driver layer, AAudio, that is even marginally adequate for musical use. Only recent devices support this, and even some of them have issues. The Oboe driver used by this app switches internally between AAudio and the older OpenSL ES audio driver to suit your device.

If you experiences glitches or low latency, try the following:

- Look at the Csound messages to see which driver Oboe is using, and use the ***Settings***, ***Audio drive***r choice to set the other one.
- Experiment with Csound's `ksmps` option in the orchestra header.
- Close all other apps on your device, and clear all caches.
- Turn on Airplane Mode.
- Experiment with your device's Power Saving or Performance mode.