User Guide
Csound for Android

Michael Gogins

31 August 2023

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.

Release Notes

Version 39

This is a bug fix release. The sizes and spacings of the buttons, labels, and sliders in the ***Widgets*** tab have been refined in order to display more widgets on smartphones.

Version 38

This is a bug fix release. The editor size was misbehaving when the mobile keyboard appeared or disappeared; fixed by changing the host element for the editor back to `<div>` from `<pre>`.

Introduction

The Csound for Android app now targets Android version 13; this is the first major release of the app since 2021. This has required adding `MANAGE_EXTERNAL_STORAGE` permission, so that the app can now read and write all files on a device. Be assured that the app does not, on its own, read any private data, or send any private data anywhere, or download any data or software on its own. Of course, a user could write a .csd file that can do these things, so exercise due care.

Android versions from 10 forward are still supported.

Upgraded libraries

- Csound has been upgraded to version 6.18.1.
- Google's Oboe audio driver wrapper has been upgraded to version 1.7.5.
- The Ace Editor has been upgraded to version 1.24.1.
- Various plugin opcode libraries have been upgraded.

Enhanced Functionality

- The widgets are now arranged with one button to the left of each slider, and the number of buttons and sliders has been increased to 12 each.
- The code editor now has visible scrollbars, faster scrolling, and a context menu for search and replace, configuring the editor, etc.

- The ***User Guide*** has been improved with instructions for using an Android emulator to run the app on notebooks or desktops.
- The data from *all* accessible sensors on a device are now routed to Csound event channels. The names of the sensors are printed to the ***Messages*** pane when Csound starts up. The names of the corresponding Csound control channels are the same but with the index of the value appended, e.g. the `accelerometer` sensor events convey three values, so its Csound control channel names are `accelerometer1`, `accelerometer2`, and `accelerometer3`.

Bug Fixes

- The app no longer crashes when run without first loading a .csd file.
- The broken link to the online ***Csound Reference Manual*** has been fixed.
- A number of plugin opcode libraries that went missing in the Android build have now been restored.
- A bug in requesting permission to write files in Android 11 and higher has been fixed.

Downgraded Functionality

- The Ableton Link opcodes have been removed.
- The LuaJit opcodes have been removed.

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 almost complete backward compatibility since then, and widely used and taught for the creation of electroacoustic and computer music. Despite 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, several phase vocoders, several sample players, a plethora of digital oscillators, filters, and envelope generators, and transparent multi-threading for higher performance of some types of pieces on multi-core systems.

The Csound for Android app is by Victor Lazzarini, Steven Yi, Michael Gogins, and others, and is currently maintained by Michael Gogins in the csound-android repository. The app contains virtually all of Csound, including many 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.

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 same builtin instance of Csound. This enables, for example, a piece to be written in HTML5 but utilize the builtin widgets, and so forth. The tabs are:

- ***Editor*** – Provides a built-in text editor based on Ace, with find and replace functionality. Touch the editor screen to enable a context menu (a little box with three dots), from which you can select the ***Find*** command.
- ***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 12 buttons, 12 sliders 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, or an HTML5 .html 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 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.
- ***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. Please note, it may be necessary to provide the full path to the Csound app's public data directory as the value of each directory, or as the first

part of each directory. This information is printed by the Csound app in the Messages pane when it first starts.

- ***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 `butt1` through `butt12`, `slider1` through `slider12`, `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”
```

The buttons send 1 when pressed, and 0 when released. The trackpad sends the `x` and `y` positions whenever the user is moving his or her finger on the trackpad. These positions are remembered when the finger is removed.

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 *Message from Another Planet* example. 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>
```

Running on Other Operating Systems

The Csound for Android app can run on any desktop operating system that has an Android emulator installed. This can sometimes provide higher performance or a better user interface than running on an actual Android device. It makes the Csound for Android app into a cross-platform, universal build of Csound.

Here I will document the use of Android Studio to create and run an Android emulator on macOS. The procedure would be much on the same on Windows or Linux. For users who do not wish to install and configure Android Studio, which is a bit of a heavyweight, I will also mention some other Android emulators.

Android Studio as an Emulator

Install Android Studio using the download, installation, and configuration instructions here.

To create an emulator, create a virtual device for Android 13 (but some versions of Android should also work). Run Android Studio. Use the **Tools** menu, **Device Manager** command to open the **Device Manager** panel. Use the **Create Device** button to create a virtual device. You can select a preconfigured device that is similar to the device you need to emulate using the **Choose a device definition** button, or you can use the **New Hardware Profile** button to open the **Virtual Device Configuration** dialog. If you plan to perform Csound pieces on the emulator, consider creating a new virtual device with the optimal display for your computer, and with as much memory as your computer can support. If the amount of memory provided does not seem sufficient, configure Android Studio itself to run the Java virtual machine with a larger heap.

To run Csound for Android:

- If your emulator supports the Google Play Store, go there in the emulator and install the app from here.
- If your emulator does not support the Google Play Store, download the .apk file from the Csound for Android GitHub repository, run the emulator, and drag and drop the .apk file on the emulator screen. It will automatically be installed.

You can run the emulator directly from a terminal, e.g. on macOS execute: `/Users/michaelgogins/Library/Android/sdk/emulator/emulator -avd Perform_Csds_API_34 -netdelay none -netspeed full`. Use the `-help` option to view all command-line options for the emulator. For example, `–memory <megabytes>` will configure the emulator's heap size.

Of course, if you use the emulator often, you can create a shell script and/or desktop shortcut to run it in one step.

Other Android Emulators

In addition to using Android Studio, it is also possible to use other Android emulators such as VirtualBox (setup for Android instructions here), NoxPlayer, or Bluestacks.

Be aware that using any Android emulator may involve security risks. However, downloading the emulator only from the official vendor's download site, scanning the download for malware, and being wary of the same things one would be wary of when running any Android app should minimize the risks to an acceptable level. After all, hundreds of millions of people use these emulators.

Examples

Built-In 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!

Issues and Workarounds

Audio Performance

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 can be configured by the user for either AAudio or 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.

File System

The Android operating system has new and sometimes awkward security restrictions on how applications can read and write files on your device.

In Android 10, applications are required to specify any storage directories on which the user might write files. For the Csound for Android app, the default public storage directory is `Music`, which on many devices is `/storage/emulated/0/Music`. If your pieces cannot read or write some file, such as a SoundFont, sample soundfile, include file, or anything else, try putting everything in this directory. Subdirectories of this should also work. On startup, you can look at the ***Messages*** pane to see what Csound is actually using for the public data directory.

From Android 11 on, security restrictions have made it more and more difficult to read and write files. For this reason, the Csound for Android app requests the `MANAGE_EXTERNAL_STORAGE` permission, which allows the app to read and write files anywhere on a device. However, rest assured that the app never, on its own, reads any private information, or sends any private information anywhere, or downloads any data or any software on its own. Of course, a user can write a Csound piece that can do these things, so exercise due care.

Screen Sizes

The sizes of fonts and widgets in the Csound for Android app can be configured to some extent in the device's Android settings, namely in the ***Settings***, ***Display***, ***Font size and style*** section and ***Screen zoom*** section.

Editor

The Ace editor used in this app currently has some issues.

- The user should touch the editor to bring up the virtual keyboard, only in text above where the keyboard will appear.
- A context menu (three dots) appears when the editor is touched. The ***Select All***, ***Cut***, and ***Find*** (and ***Replace***) commands work, but the ***Paste*** and ***Undo*** commands do not work. However, the standard Android select, copy, cut, and paste operations *do* work.
- The scrollbars operate in a manner that might not be intuitive for desktop users. The visible bars at the left and bottom of the editor are the scrollbar "thumbs", and on a mobile device they are not draggable except as part of the editor panel. Rather, the thumbs simply indicate the relative size and position of the visible document with respect to the entire document. To scroll, simply touch a scrollbar (the very leftmost or bottommost part of the screen, where the thumbs appear) and the editor will jump to the corresponding position in the document.

HTML Pieces

When an HTML example loads, the app may briefly display a popup message about a missing `csound.js` file. This message can be ignored and should not recur. The piece should render properly anyway.

If the HTML for the piece is not rendered correctly, it may help to exit the app and reload the piece.

In general, for HTML pieces, it is necessary to click on the ***Run*** button, as for any other Csound piece, before Csound is instantiated in the JavaScript context and can render the piece.

Debugging Crashes

If Csound crashes when it starts, or while it is running, then the Csound for Android app simply exits without warning or information.

That is because Csound is written in C, and the C runtime library handles crashes using signal handlers or an atexit function. But the Csound for Android app is written in Java, and runs on a Java virtual machine. When a fatal error occurs, the signal handler is duly called, but the state of the Java virtual machine at that time cannot be known. Therefore, the results of calling Java code from the signal handler cannot be determined. And so, it's not possible to throw a Java exception or to call from the signal handler back into Java to show an alert from the signal handler.

However, it is still possible to figure out what caused the crash. This is done by putting a diagnostic print statement such as `prints “Got this far…\n”` in your Csound orchestra code, starting in the orchestra header, and moving this print statement around in the orchestra until the app crashes before you see the message. That must be just after where the crash happened, so you can often figure out the cause of the crash by the carefully reading the code that runs just before the unprinted print statement.