CSOUNDO FOR ANDROID

Rory Walsh,
Department of Music and Creative Media,
Dundalk Institute of Technology
Dundalk, Ireland
rory.walsh@dkit.ie

Conor Robotham,
Department of Music and Creative Media,
Dundalk Institute of Technology
Dundalk, Ireland
conorrobotham@gmail.com

ABSTRACT

The Csoundo library combines the sound compiler Csound with Processing, a java-based programming language for visual applets. The combination of the two provides a framework for the rapid development of interactive audio-visual applications. Csound and Processing have recently been ported to the Android platform so these languages may now be used to develop applications directly for Android devices. However, at present, Csound for Android has only been implemented with basic Android Graphical User Interface(GUI) elements while Processing for Android lacks support for external libraries, including its established audio libraries. Csoundo for Android bridges the gap between the two languages and allows developers to rapidly build interactive audio applications without having to be concerned about the inner-workings of Android development.

1. INTRODUCTION

The mobile devices market has been booming exponentially year on year, and with this has come an increased interest in not only the development of 'apps' themselves, but in the development of IDEs for various mobile platforms. Mobile devices offer new forms of interactivity including multi-touch screens and various gestural sensors and accelerometers. The Android Platform, developed by Google, is a huge player in this market and, unlike Apple, it provides developers with the opportunity to develop and distribute applications freely.

In recent years Csound[1] has been ported for use with a number of mobile devices. At the same time Processing[2] is continuing development on Android compatibility and now provides an interface to the latest Android Application Programming Interface(API). The Csoundo[3] library caters for the calling of Csound methods from a Processing program or so-called 'sketch' on a desktop computer. With the aforementioned ports in place, an upgrade of Csoundo for development on Android devices seemed like the next obvious step in the life-cycle of the library. It provides what is likely the easiest method of developing an interactive Android audio-visual application. Not only that, but whatever application is built using Csoundo will also function on a desktop computer in the form of a Java applet. This is ideal for testing because it means one can develop apps free of having an Android device. The following sections outline the various frameworks used in the development of Csoundo for Android.

1.1. Csound for Android

In early 2012, Csound was ported to the Linux-based Android operating system[4]. The Android Native Development Kit (NDK) was employed to build a native shared library, libcsoundandroid.so, composed of libcsound, libcsnd and libsndfile. libcsound consists of the main Csound library; libcsnd contains the SWIG-wrapped interface extensions as SWIG exports, to be called by SWIG functions within JAVA classes; and lastly libsndfile accommodates a range of different audio formats.

The native shared library must be called through JAVA for the Android Dalvik compiler, as this is the main API language for the Android OS. Therefore, the developed Java API used on this system comprises of the wrapped Csound library, a modified version of the Csound interfaces library used for common method calls, and some unique value cacheable classes which simplify some of the Android-specific functionalities such as the accessing data from accelerometers.

For audio IO on Android, Csound Android offers two distinct possibilities. The first uses the standard AudioTrack interface, while the second, default mode, employs the use of the OpenSL which replaces the usual Csound IO modules such as portaudio, alsa and jack[4].

1.2. Processing

Processing is a graphics generation language written in Java that allows users to produce illustrative or interactive programmes. It comes with a choice of rendering engines, and as of Processing 2.0, OpenGL has now been integrated into the core of Processing. The benefits of which are that sketches can now be exported far easier to a wide variety of mobile devices. Processing is developed as a "software sketchbook" to assist in the rapid development of artistic concepts. Each 'sketch' contains a setup and draw method which are used to initialize data and continually update it based on a frame rate respectively. In 2010 Processing was ported to Android. The upshot of this port is that users can now easily swap between Android Mode and Java Mode without having to rewrite any of their code.

1.3. Csoundo

The original Csoundo library for Processing was developed by Jacob Joaquin and released in the fall of 2010[7]. Csoundo takes advantage of the Csound class provided by the wrapped Java Csound library. In doing so it provides methods to compile and send/receive messages to and from Processing from Csound. Importing the Csoundo library and creating an instance of a Csoundo object provides the developer with access to a range of methods that make calls to the underlying Csound class. Events can be triggered from the Processing environment by sending values to an instance of Csound via the software-channel bus. At the time of Csoundo's original development Csound had yet to be ported to mobile devices. For the authors of this paper, extending the Csoundo library to take advantage of these latest Csound developments seemed like the next logical step in the library's life-cycle.

1.4. Other notable Android audio projects

It would be remiss of the authors at this point not to mention some other notable audio for Android projects, in particular libpd[5] and PureDataP5[6]. libpd is a Pure Data library for mobile devices. It frees Pd of GUI dependencies and lets users build their own interfaces using native methods while providing them with all the audio processing power of Pure Data. PureDataP5 combines libpd with Processing, allowing users to take advantage of both frameworks. As of the time of writing this paper, users wishing to use Processing and libPd for the development of Android applications must use the Eclipse development environment in order to deploy their apps to their Android device. This is not the case with Csoundo for Android.

2. CSOUNDO FOR ANDROID

The following section provides an overview of the Csoundo for Android library, beginning with a look at how Processing libraries are organised and structured. Following this is a description of the Csoundo class itself, and, finally, the steps involved in modifying the Processing framework to accommodate external libraries are outlined.

2.1. Processing Libraries

Libraries for Processing are quite straightforward to construct. Each library must be housed in a self contained folder that must be placed in the Processing sketchbook 'libraries' folder. When Processing is run for the first time it will create a default 'sketchbook' folder within one's home directory. Within each library folder there will be a directory called 'libraries' which will contain the relevant Java archive files for the Processing library. In the case of Csoundo there are four .jar files in the libraries folder:

Table 1: *Libraries used in Csoundo*.

Java Archive	**Features**
Csnd.jar	Java interface to both libcsound and libcsnd. Needed for running Csoundo sketches on a Desktop PC.
CsoundAndroid.jar	Android interface to Csound API, also contains unique Android utility methods. This library is needed for running Csoundo sketches on Android.
Android.jar	Interface to native Android methods.
Csoundo.jar	Simplified Processing wrapper for functions found in csnd.jar

2.2. The Csoundo class

Csoundo is the main class used in developing Csound sketches with Processing, both in desktop and Android mode. The overloaded Csoundo constructors are defined as,

```
  public Csoundo(PApplet theParent, Context context)
  public Csoundo(PApplet theParent, String _csd)
```

The first constructor creates an object of type CsoundAndroid(), while the second creates an object of type Csound(). You'll note that the constructor for setting up a CsoundAndroid class gets passed an object of type Context. This object is very important as it provides access to some useful application-specific resources and classes. It is through this object for example that we can retrieve the full path to the .csd file to be compiled by Csound. The one drawback to having overloading constructors like this is that users will need to use one or the other depending on their target platform. More eloquent ways of setting up Csound are being investigated and it is hoped that a single multi-target interface can be developed in the future.

The Csoundo constructor is the only overloaded method, which means that all the other Csound API methods exposed by Csoundo are the same whether the user is developing for Android or desktop applications. Below is a list of most useful methods found in the Csoundo:

Table 1: *Useful Csoundo methods*.

Method	**Summary**
`Csoundo(...)`	Main constructor
`get0dbfs()`	Returns the 0dbfs value, set in the orchestra header.
`getChn(...)`	Returns the value of the specified chn bus.
`setChn(...)`	Sets the value of the specified chn bus.

`getPerfStatus(...)`	Returns the status of the Csound Performance Thread.
`ksmps()`	Returns the ksmps, samples per k period
`nchnls()`	Return the number of channels
`tableGet(...)`	Returns a value from a table, determined by index
`tableSet()`	Sets the value of a particular index in a Csound table.
`SetOptions(...)`	Overwrites CsOptions.

2.3. Modifying Processing

When users hit the 'Send to Device' button in Processing, Processing will export all files needed to a temporary directory before it pushes them across to the attached Android device. In the case of Csoundo all the Csoundo Java files get distributed to this temporary directory before being sent to the device. The one problem is that as of the time of writing Processing does not support external libraries when in Android development, so while Processing specific libraries are supported, external Android libraries are not. This represented a problem for Csoundo as it is completely dependant on the presence of a Csound library, be that csnd.jar in desktop mode, or CsoundAndroid in Android mode. As a result some simple modifications have had to be made to the Processing Java archive file so that external dependencies get referenced and included. The AndroidBuild.java class file contained within the pde.jar archive is where all the modifications took place. This class contains all of the functions needed by Processing in order for it to export an entire project to a temporary Android build folder on disk before sending it to a device. In order to solve the issue of external libraries, the writeProjectProps() method needed to be slightly altered. This method is responsible for writing the project properties files for the exported applications. In order for CsoundAndroid to be pushed to the device it was necessary to add the path to the CsoundAndroid folder via the project properties Android.Library.Reference identifier. On Linux systems the CsoundAndroid folder should be placed in ~/sketchbook so that Processing can find it when it comes to exporting as an Android app.

The next modification to the AndroidBuild class was to its writeRes(File resFolder, String className) method. This needed to be changed so that it would export the Csound file associated with the Csoundo sketch, which is located by default in the sketches 'data' sub-folder, to the Android package's res/raw folder. Once in the resources raw folder the .csd file can be accessed using the Android context method openRawResource(). Once the file has been accessed, Csoundo will create a temporary copy of the file in a known location, which can then be passed to the Csoundo constructor in order to be compiled by Csound.

The final modifications to AndroidBuild was to the copyLibraries(File libsFolder, File assetsFolder) method. This method copies all the relevant Processing libraries used in one's sketch to the Android build folder's lib directory. This directory will contain processing-core.jar and whatever other Processing libraries your sketch uses. A conditional test was placed within this function to see what the export target platform is. If the target platform is Android then there is no need to export csnd.jar as the functions exposed in this library are all available in the CsoundAndroid archive. Not only does this step prevent the wast of valuable memory on one's Android device, it also prevents a series of messy compiler errors when porting your applications to Android.

3. GETTING SETUP WITH CSOUNDO FOR ANDROID

In order to use Csoundo for Android you will need to have the latest versions of both Csound and Processing downloaded and installed on your desktop. Any recent version of Csound5 will work fine, but you'll need to use version 2.0b.7 of Processing, which as of the time of writing is the most current beta release of Processing 2.
Following this you will need to install the Android SDK. Rather than downloading the entire SDK it is better, and far quicker to simply download the SDK manager and install the following packages:

- Tools/Android SDK Tools
- Tools/Android SDK Platform-tools
- Android 2.3.3(API 10) / SDK Platform
- Android 2.3.3(API 10) / Google APIs

Users of 64-bit versions of Linux should also install 32-bit compatibility libraries using the following command:

```
sudo apt-get install ia32-libs
```

Once you have Processing working in Android mode you can grab the latest source code for Csoundo from the following website:
https://github.com/rorywalsh/Csoundo
Information on how to install Csoundo is provided on the aforementioned github repository.

4. EXAMPLES

The following section presents two simple Csoundo sketches. The first shows a simple sketch which illustrates how Processing events can be used to trigger sound events in Csound. The second shows how the opposite is possible, i.e., how to trigger graphical events in processing from Csound. Please note that these examples were written to be as simple and succinct as possible. For more interesting and involved programs please check out the Csoundo examples directory.

4.1. From Processing to Csound

This simple sketch uses the users finger position to control a simple random event to draw lines to screen, while turning each into a note being played with Csound. The code for the basic Processing sketch is given below.

```
Example 1.pde
import csoundo.*;
Csoundo cs;

void setup(){
size(displayWidth, displayHeight);
frameRate(10);
smooth();
cs=newCsoundo(this,
              super.getApplicationContext());
cs.run();
}

void draw() {
cs.setChn("mod", mouseX);
cs.setChn("car", mouseY);
}
```

The sketch's draw() method will get called 10 time per second, and each time it is called the X/Y coordinates of the mouse, or the users finger in Android mode, will get sent to an instance of Csound. The data will be transferred on two named channels, "mod", and "car". The Csound instrument is defined in the following code.

```
Example 1.csd
<CsoundSynthesizer>
<CsOptions>
-odac -+rtaudio=null -d -m0 -b512
</CsOptions>
<CsInstruments>
sr = 44100
ksmps = 64
nchnls = 2
0dbfs = 1

gaSig init 0

instr 1
kmod chnget "mod"
kcar chnget "car"
amod lfo .1, kmod
acar oscil amod, kcar, 1
gaSig = acar
endin

instr 2
acombL comb gaSig*.1, 2, .3
acombR comb gaSig*.1, 2, .42
outs acombL*2, acombR*2
endin

</CsInstruments>
<CsScore>
f1 0 1024 10 1 0.5 .2
i1 0 [60 * 60 * 24]
i2 0 [60 * 60 * 24]
</CsScore>
</CsoundSynthesizer>
```

The most important parts of the above code are the two lines using the *chnget* opcode. *chnget* is passed the name of the channel on which to listen for data from Processing. Finally, the Csound score tells the two instruments to run for 24 hours, which should be long enough for a single user session.

4.2. From Csound to Processing

The major issue with triggering graphical events in Processing with Csound is that both system use completely different sample rates. Therefore a lot of data being sent and accessed through the chnget/chnset opcodes will be lost in transmission. To get around this issue users can use a callback system that will trigger Processing events each time Csound sends a message on a particular channel. In order to do this we must set up a callback method within our Processing sketch. The channel callback method must use the following method prototype:

```
public void outvalueCallback(String chan,
double val)
```

Once you have defined this method it will be called every time Csound updates a channel. Because the chnget/chnset opcodes do not work via callbacks, users will need to use the outvalue opcode instead. When using the out/in value opcodes, a registered callback will be triggered each time an out/invalue opcode is called from within Csound. Users can then tell Processing to draw something only when a channel message has been updated.

```
Example 2.pde
import csoundo.*;
Csoundo cs;
float xPos=0, yPos=0, ballSize=0;

void setup(){
//Java Mode
cs = new Csoundo(this,"Example2.csd");
cs.run();
background(0);
size(300, 300);
smooth();
noLoop();
frameRate(10);
strokeWeight(5);
}

void draw(){
fill(ballSize*255,(xPos*width)*255,
                  (yPos*height)* 255);
ellipse(xPos*width,yPos*height,
                   50*ballSize+10,
                   50*ballSize+10);
fill(0, 0, 0, 10);
rect(0, 0, width, height);
}

public void outvalueCallback(String chan,
                             double val){
if(chan.equals("xPos") == true)
  xPos = (float)val;
else if(chan.equals("yPos") == true)
  yPos = (float)val;
else if(chan.equals("ballSize") == true)
  ballSize = (float)val;
redraw();
}
```

The corresponding Csound instrument looks like this:

```
Example 2.csd
<CsoundSynthesizer>
<CsOptions>
-+rtmidi=alsa -m0d -o dac:hw:1,0
</CsOptions>
<CsInstruments>
sr = 44100
ksmps = 1024
```

```
nchnls = 2
0dbfs = 1

instr 1
kFreq randh 1000, 10, 2
kAmp randh 1, 10, 2
kHarms randh 4, 10, 2
kTempo init 1
kGo metro abs(kTempo)+1

if(kGo==1) then
event "i", 10, 0, 3, kAmp, kFreq, kHarms
kTempo randh 5, 100, 2
endif
endin

instr 10
outvalue "xPos", (6-abs(p6))/6
outvalue "yPos", abs(p4)/1
outvalue "ballSize", (1000-abs(p5))/1000
aexp expon abs(p4)+.1, p3, 0.001
a1 buzz aexp/4, p5, p6, 1
outs a1, a1
endin

</CsInstruments>
<CsScore>
f1 0 1024 10 1
i1 0 3600
</CsScore>
</CsoundSynthesizer>
```

It is important to set your control rate to a relatively low value, otherwise Processing will not be able to keep up with the high frequency of updates and you may experience some audio drop-outs. In the above code ksmps is set to 1024, which gives a control rate of 43Hz. This essentially sets Processing's max frame rate to 43, some 10 or so frames higher than standard video.

5. CONCLUSION

The latest update to the Csoundo library has made development of Csound sketches with Processing a very attractive option for anyone looking to create audio-visual applications for Android. On top of that it provides, for the first time, a truly integrated development environment for Csound users wishing to develop applications for Android devices. If users wish to develop standard interfaces to their apps they can do so using one of Processing's GUI control libraries such as controlP5, which provides an array of standards controls such as sliders, buttons, xypads, etc. At the time of writing, all other development toolkits for Android involve the use of an integrated development environment such as Eclipse, which puts them beyond the reach of anyone new to coding computer applications. However, Csoundo provides a far easier and more streamlined approach to developing applications for Android.

It must also be noted that the Android 'mode' for Processing 2.0 (beta) is still very much under development, and it is not yet clear if the developers will offer this development mode as standard, or as an optional add-on, or as a completely separate package. Until this becomes clear it is somewhat difficult to predict how Csoundo will be packaged in the future. One option being explored is a single Csoundo package that includes not only the necessary JAVA archives for Csound, but the complete Android development mode for Processing. Another option is to add a new development mode, called 'Csoundo' to Processing, but the maintenance of such a mode might prove to be somewhat prohibitive. Whatever form Csoundo for Android eventually takes, it's of the utmost importance that it be as simple as possible for end-users to set up and get running.

Full source code, code examples and necessary libraries for the current version of Csoundo are available at:

https://github.com/rorywalsh/Csoundo

6. AKNOWLEDGEMENTS

Thanks goes to everyone on the Csound and Processing forums, in particular Steven Yi whose insights and direction proved to be invaluable to this project. The authors also wish to express their sincere thanks to Jacob Joaquin. None of the developments discussed in this paper would have come to fruition had it not been for his willingness to share his work on the original Csoundo library with the wider open source community.

7. REFERENCES

[1] ffitch, J. 2005. *On The Design of Csound 5*. In: LAC 2005 Proceedings: 3rd International Linux Audio Conference, April, 2005, Karlsruhe. Karlsruhe: Zentrum fur Kunst und Medientechnologie, pp. 37-42.

[2] Reas, C. and Fry, B. 2007. *Processing: a programming handbook for visual designers and artists*. Cambridge:MIT Press.

[3] Joaquin, J 2010. *Csoundo — A Csound library for Processing*. [online] 2010, available: https://github.com/jacobjoaquin/Csoundo [accessed 24 Jan 2013].

[4] Lazzarini, V., Yi, S., Timoney, J., Keller, D. and Pimenta, M. 2012. *The Mobile Csound Platform.* In: Marolt, M., Kaltenbrunner, M. and Ciglar, M. eds., Proceedings of the International Computer Music Conference, September, 2012, Ljubljana. San Francisco:

[5] Brinkmann, P. 2011. libpd, [online], available: https://github.com/libpd/ [accessed 29 Jan 2013

[6] Esler, R. (2012). Libpd with Processing, Robert Esler, [online], 20 January, available: http://robertesler.com/libpd-with-processing/[accessed 3 August 2012].

[7] Joaquin, J. (2010). Announce: Csoundo Processing Library, [online], 9 August, available: http://csound.1045644.n5.nabble.com/Announce-Csoundo-Processing-Library-td2268382.html/[accessed 13 December 2012].