Csound for Android

Victor Lazzarini, Steven Yi and Martin O'Shea

December 1, 2014

1 Introduction

Welcome to Csound for Android! This document will discuss the details about using Csound for Android OS using the Eclipse IDE.

For those with knowledge of Csound, hopefully you will see that the value of your knowledge is only enhanced by offering a new platform on which to create musical software and works.

1.1 Regarding the Csound for Android Examples Project

This documentation covers discussion of the Csound for Android API. Users interested in diving in to see how the API is used may want to download the Csound for Android Examples Project which contains a set of examples that cover different use cases of the API. The code for each example may be useful in giving you a jump-start in building your own application.

1.2 Regarding the LGPL License

The Csound for Android includes Csound and libsndfile. These are distributed as static libraries. Users of the Csound for Android API must comply with the licensing requirements of the GNU Lesser General Public License v2.1, which both libraries use. Please carefully review the license files that accompany each project (you can view a generic version of the LGPL v2.1 license at http://www.gnu.org/licenses/lgpl-2.1.html).

2 Getting and Using the Csound for Android API

The Csound for Android library is distributed as a Zip release from the Csound Sourceforge page. The Zip archive includes:

- Statically compiled libcsound.a and libsndfile.a, compiled as universal binaries for armv6, armv7, and i386 CPU architectures (to work with both Android devices and simulators)
- C Headers for the Csound C API
- Java API source
- Documentation

You will also need to install the android NDK available at AndroidDeveloper site, to compile the native code in Csound for java.

Csound for Android was chosen to be delivered as pre-compiled libraries and headers for easy inclusion into projects. After starting a new project in Eclipse, do the following:

1. Import the CsoundAndroid Library into Eclipse using file>import from the location you downloaded it to.
2. Right-click your project and from the context menu choose “Properties”
3. In the window that opens, select the "Android tab" on the left hand-side of the window, go to the right of the box labelled "library" select "add". Select the "CsoundAndroid Library" which you have just added to your workspace(Step 1).
4. If at anytime when working with the CsoundAndroid library the project does seem not to recognise the CsoundAndroid is added, select "Project" from the menu bar and then select "Clean" to rebuild the paths in the project.

After adding the CsoundAndroid library, it should now be a part of your project. You will now be able to reference both the standard Csound C API as well as the Java CsoundObj API from your project code.

3 Introduction to the API

3.1 CsoundObj and Csound API’s

Csound for Android is released with the standard Csound C API as well as a custom Java CsoundObj API that has been designed to make developing on Android OS convenient. The CsoundObj API includes methods for

binding widgets to channels (used to communicate to and from Csound), enabling hardware sensors, and more. For further detail, please consult the CsoundObj.java class.

While the CsoundObj API has been designed to ease things for Android OS development and to follow conventions of Java, the decision was made not to wrap everything in the Csound C API. . The Csound java Object that a CsoundObj class holds can be accessed via the **getCsound** method in CsoundObj. For more information about the Csound C API, consult Csound.java class.

4 Using the CsoundObj API

The CsoundObj API revolves around the Java CsoundObj class. This class contains a Csound Object and has methods for running Csound, as well as convenience methods to help aid developers in connecting elements to Csound. By itself, CsoundObj can take in a CSD file and render it. By using CsoundBindings, objects can interact to read values from and write values to Csound. Beyond that, extended features can be accessed by using the Csound C API together with the Csound object.

4.1 Designing Csound CSD projects to work with Hosts

To communicate to and from a host program with Csound, you will most likely use **chnset** and **chnget** opcodes. These opcodes will allow you to read from and write values to a named channel. Your host program will also be writing to and reading from these same channels. As a byproduct of using named channels, your CSD will be portable to work on other platforms (Desktop, IOS); porting over apps to/from Android then will only involve redoing the application and UI code, while your audio engine (Csound) should "just work."

4.1.1 CsOptions

Csound will use the CsOptions you provide. A standard set of flags to use are "`-odac -dm0 -+msg_color=0`". This will tell Csound to play to dac in realtime, turn off displays, use message level 0, and turn of message colors. This will minimize messages going out to the console or to your message handler if you set one. You may also need "`-iadc`" if your project uses audio input. (You can always consult the Csound Examples project to see what CsOptions are used in the CSD's for that project.)

Note: In Csound5 for Android, it was previously required to set "`-+rtaudio=null`" when using audio. This is no longer required for Csound 6 for Android.

4.2 CsoundBinding for Communicating with Csound

The CsoundObj API has been created to ease communication with Csound by using objects that implement the CsoundBinding interface. The interface definition is as follows:

Listing 1: CsoundBinding interface Definition

```
public void setup(CsoundObj csoundObj);

public void updateValuesToCsound();

public void updateValuesFromCsound();

public void cleanup();
```

CsoundBindings are used to both read values from Csound as well as write values to Csound. The lifecycle of CsoundBindings is as follows:

- **setup** - this has now been changed. This method is called after Csound's compile call but before the main performance loop. CsoundBindings should use this method to cache any channel pointers and any other values they will need during performance.
- **updateValuesToCsound** and **updateValuesFromCsound** - these methods are called during the Csound performance loop. **updateValuesToCsound** is called before each call to csoundPerformKsmps, while **updateValuesFromCsound** is called after each call.
- **cleanup** - this method is called after Csound has completed its run and should be used by CsoundBindings to free up any allocated data and remove references to channel pointers.

By using CsoundBindings, CsoundObj functionality can be extended to communicate with as many items as you would like. The Csound for the Android API contains pre-made wrapper classes for common UI classes (Slider, Button) as well as hardware sensors (Accelerometer). CsoundObj has helper methods for the CsoundBindings that come with the CsoundObj API, as well as the generic **addBinding** and **removeBinding** methods for

adding custom CsoundBindings. Please consult these classes as well as their use in context within the Csound for Android Examples project.

5 Common CsoundObj API Methods

5.1 Binding Widgets to CsoundObj

The CsoundObj API contains the following methods for binding widgets:

Listing 2: Methods for Widget Binding

```
public CsoundBinding addButton(Button button, String
    channelName))
public CsoundBinding addButton(Button button, String
    channelName,int type)
public CsoundBinding addSlider(SeekBar seekBar, String
    channelName, double min, double max)}

These methods allow for easy binding of UISeekBars, and
    UIButtons, and return the CsoundBinding that was created
     to wrap the widget. If the design of your app requires
    that you remove a widget from being used with CsoundObj,
     you can use the returned CsoundBinding and pass it to
    the removeCsoundBinding method. To bind your own custom
    widgets, you will need to create your own CsoundBinding.
      There are examples of both using the convenience
    widget binding methods as well as custom CsoundBindings
    in the Csound for Android Examples project.
```

6 Interfacing with Hardware

6.1 Audio Input and Output

CsoundObj has been designed to connect everything necessary for audio input and output from Csound to *(*OpenSL) in the when the CsoundLib Library is compiled in with the NDK, the CsoundObj can also interact with audio track at run time (this is to allows for downward compatibility with 2.2 and lower). Enabling input and output depends on what commandline arguments are given when running Csound. The commandline arguments should be supplied as part of the CSD's **CsOptions** section. To enable audio output, use **-o dac** and to enable audio input, use **-i adc**.

6.2 Accelerometer

CsoundObj has built-in support for use of the accelerometer found in android devices. CsoundObj has the following methods to enable these features:

Listing 3: CsoundObj Accelerometer Method

```
public CsoundBinding enableAccelerometer(Context context)
]
```

When the feature is enabled, CsoundBindings that wrap the sensor will send values into Csound via hardcoded channels:

- Acclerometer
  - accelerometerX
  - accelerometerY
  - accelerometerZ

Once a sensor has been enabled, you can access those values in Csound using **chnget**. For further study, please see the *Hardware Test* example in the Examples project.

7 Csound for Android Examples

The Examples project contains a number of simple examples that illustrate different aspects of working with Csound for Android. The following is a brief description of each of the examples.

7.1 Simple Test 1

Simple example that plays ascending notes. The pitch of the notes can be altered by using the SeekBar. Also, a ToggleButton is used to turn on/off the rendering of Csound. In the code, you'll find that the callback that is connected to the ToggleButton shows the basic usage of CsoundObj to render a CSD that is included as a resource for the project:

Listing 4: Example code showing configuring and starting a CsoundObj

```
public void onCheckedChanged(CompoundButton buttonView,
    boolean isChecked) {
         if(isChecked) {
           String csd = getResourceFileAsString(R.raw.test);
           File f = createTempFile(csd);
```

```
            csoundObj.addSlider(fSlider, "slider", 0.,
                1.);
            csoundObj.addListener(SimpleTest1Activity.this);
            csoundObj.startCsound(f);
          } else {
            csoundObj.stopCsound();
          }

        }
      });
```

7.2 Simple Test 2

This is a generative music example that contains a number of SeekBars that affect the rate of notes generated, the duration of notes, and the ADSR envelope for each note.

7.3 Button Test

This example uses a CSD based on the one used for Simple Test 2, but depends on the user to trigger a button to generate each note. The two buttons in this example show two different ways in which to integrate buttons with CsoundObj:

1. Using CsoundObj's **addButton** method, which will setup a k-rate channel for Csound, the value will be 0 when the button is not pressed, and will be 1 for one ksmps period when a button is pressed (it returns to 0 the following ksmps period).

2. Using a standard button callback, the callback will create a string score and send that to Csound using CsoundObj's **sendScore** method. (See code below.)

Listing 5: Example code showing sending score text to CsoundObj

```
String event = String.format("i2 0 %f", value);

csoundObj.sendScore(event);
```

Note that the second method will read the value from the duration slider when creating the note, while the first method handles reading the duration from the channel within the CSD code.

7.4 Ping Pong Delay

This example shows processing audio input in realtime, using a Ping Pong Delay. The use of audio input is controlled by the standard Csound flag **-i adc** that is found in the CSD's **CsOptions** section.

7.5 Harmonizer

This example highlights the same techniques as the Ping Pong Delay, but shows the use of Csound's streaming Phase Vocoder to create a harmonizer effect.

7.6 Accelerometer

This example shows the use of the accelerometer sensor. For this example, the accelerometer is enabled and values are read by the CSD to affect the pitch of a vco2 oscillator, as well as cutoff and resonance of a moogladder filter.

7.7 Csound Haiku 4

Csound Haiku is a generative art music work by Iain McCurdy. Number 4 from this set of pieces was chosen to exercise what this platform is of capable.

7.8 Multitouch XY

This example demonstrates a multitouch performance surface. The multitouch code maps each touch down and up to a note on and off. It also sends continous x and y values to Csound. The Csound programming in the CSD shows a technique for doing individual per-note control data mapping by dynamically assigning what channels of data each note should read from.

7.9 Waveview

This example demonstrates using a CsoundBinding to read an f-table from Csound and displaying that table. Note that the WaveView's code is doing some optimization to check if it has already loaded. It is also checking that the table itself has completed loading before trying to grab any values for the table.

Listing 6: Waveview code demonstrating reading f-tables from Csound

```
public void updateValuesFromCsound() {
  if (!tableLoaded) {
    Csound csound = csoundObj.getCsound();
    CsoundMYFLTArray table = new CsoundMYFLTArray();
    int length = csound.TableLength(1);
    csound.GetTable(table.GetPtr(), 1);
    tableData = new double[length];

    for (int i = 0; i < length; i++) {
      tableData[i] = table.GetValue(i);
    }

    tableLoaded = true;

    new Thread() {
      @Override
      public void run() {

        int width = getWidth();
        int height = getHeight();
        int middle = height / 2;

        points = new int[width];

        int tableLength = tableData.length;

        for (int i = 0; i < width; i++) {
          float percent = i / (float) (width);
          int index = (int) (percent * tableLength);
          points[i] = (int) ((tableData[index] * middle)
              + middle);
        }

        postInvalidate();

      }
    }.start();
  }
}
```

8 Making Your First Csound Android App

This section will walk you through your first implementation of the Csound library for Android:

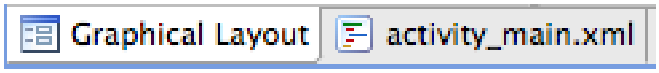
- assuming you have added the library to your project as explained in

the *Getting and Using the Csound for Android API* section; and

- that your application is called *First_Csound_App* with the package name of *com.example.first_csound_app*; and
- you used the default names for your first class and xml file, which are*MainActivity.java* and *activity_main.xml* respectively

First let us create the user interface. This is simplest using xml files, though this can be achieved through Java.

First navigate to the *res* > *layout* >*activity_main.xml* in the Package explorer in Eclipse. At the bottom left of the the xml view pane you will see a graphical layout tab and an activity_main.xml tab

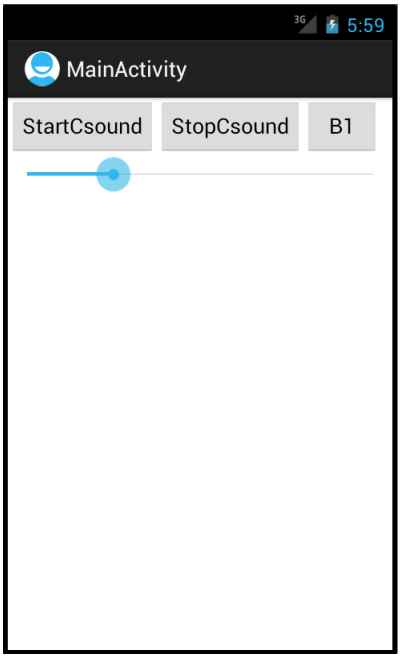


Insert the following code in the xml file. This will create three Buttons and a SeekBar as shown.

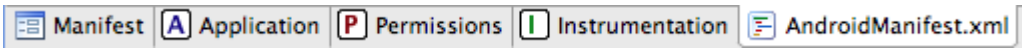
Listing 7: Code to create UI

```
<RelativeLayout xmlns:android="http://schemas.android.com/
    apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <!--This is a comment! the following inserts a Button
        and is
    soley responsiple for how the Button looks in this app.
    The id section is used for refferencing to the Button
        throughout
    the rest of the project -->

    <LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/
            android"
        android:id="@+id/ButtonList"
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        >
    <Button
        android:id="@+id/StartCsound"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:layout_centerHorizontal="true"
        android:text="StartCsound"
         />
    <Button
        android:id="@+id/StopCsound"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:text="StopCsound"
         />
    <Button
        android:id="@+id/Button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:text="B1"
         />

    </LinearLayout>

    <!--This is a comment! the following inserts a SeekBar (
        comminly reffered to as a slider) and is
    soley responsiple for how the Button looks in this app.
    The id section is used for refferencing to the SeekBar
        throughout
    the rest of the project-->
    <SeekBar
        android:id="@+id/SeekBar1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_below="@+id/ButtonList"
        />

</RelativeLayout>
```



Next go to your project manifest to add the necessary permissions for fully using the CsoundObj. This is done by selecting AndroidManifest.xml from the package explorer, then selecting *AndroidManifest.xml* from tabs at the bottom of the main pane



Adding the following to your Manifest

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.first_csound_app"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="15" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/title_activity_main" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
<!-- Add the following code-->
<!-- MODIFY_AUDIO_SETTINGS is needed to use audio effects such as environmental reverb -->
     <uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
</manifest>
```

Now that the UI is built and the manifest has been prepared, proceed to opening the main activity of the project *src>com.example.first_csound_app>MainActivity.java*. You should see the following when you open the class (depending on your Eclipse preferences the line numbering may not appear)

```
1  package com.example.first_csound_app;
2
3  import android.os.Bundle;
12
13
14 public class MainActivity extends Activity {
15
16     @Override
17     public void onCreate(Bundle savedInstanceState) {
18         super.onCreate(savedInstanceState);
19         setContentView(R.layout.activity_main);
20     }
21
22     @Override
23     public boolean onCreateOptionsMenu(Menu menu) {
24         getMenuInflater().inflate(R.menu.activity_main, menu);
25         return true;
26     }
27
28
29 }
30
```

First add the *BaseCsoundActivity.java* class to your project. This can be done by copying it into the *src* folder of your project on your computer from the CsoundAndroidExamples project that comes with the Csound for Android download.

Next you need to import the necessary packages into your package for the application to compile correctly. These include the Csound package and widget packages for the UI. To see all the automatically imported packages click the + button beside the import shown in the last image

```java
package com.example.first_csound_app;

import java.io.File;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.support.v4.app.NavUtils;
//THIS IS A COMMENT!// The following are the packages you need to import///
import android.widget.Button;
import android.widget.SeekBar;
import com.example.first_csound_app.R;
import com.csounds.CsoundObj;
import com.example.first_csound_app.BaseCsoundActivity;
////////////////////////////////////////////////////////////////////////////


public class MainActivity extends BaseCsoundActivity {


    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }


}

```

Next extend your class with BaseCsoundActivity instead of Activity (note that BaseCsoundActivity extends Activity). Then add your variables and then initialise them in the onCreate method as shown below.

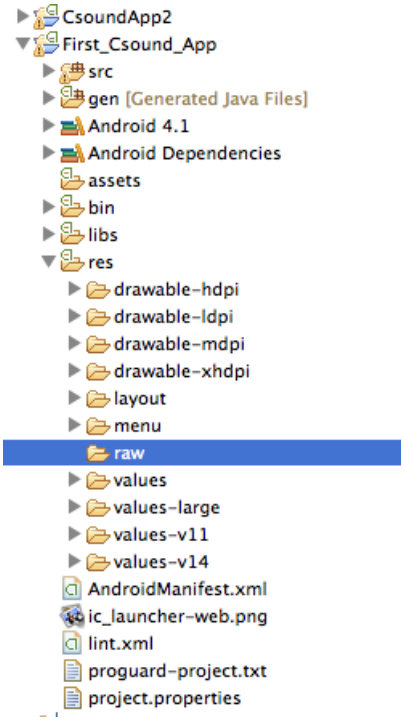
```
1  package com.example.first_csound_app;
2
3  import java.io.File;
4
5  import android.os.Bundle;
6  import android.app.Activity;
7  import android.view.Menu;
8  import android.view.MenuItem;
9  import android.view.View;
10 import android.support.v4.app.NavUtils;
11 //THIS IS A COMMENT!// The following are the packages you need to import///
12 import android.widget.Button;
13 import android.widget.SeekBar;
14 import com.example.first_csound_app.R;
15 import com.csounds.CsoundObj;
16 import com.example.first_csound_app.BaseCsoundActivity;
17 ////////////////////////////////////////////////////////////////////////////
18
19
20 public class MainActivity extends BaseCsoundActivity {
21
22     private CsoundObj csoundObj;
23     Button startCsound, stopCsound, button1;
24     SeekBar seekBar1;
25
26     @Override
27     public void onCreate(Bundle savedInstanceState) {
28         super.onCreate(savedInstanceState);
29         setContentView(R.layout.activity_main);
30
31         csoundObj = new CsoundObj();
32         startCsound = (Button) findViewById(R.id.StartCsound);
33         stopCsound = (Button) findViewById(R.id.StopCsound);
34         button1 = (Button) findViewById(R.id.Button1);;
35         seekBar1 = (SeekBar) findViewById(R.id.SeekBar1);
36
37     }
38
39     @Override
40     public boolean onCreateOptionsMenu(Menu menu) {
41         getMenuInflater().inflate(R.menu.activity_main, menu);
42         return true;
43     }
44
45
46 }
47
48
```

Finally in our Activity class we have to connect our UI comments to our CsoundObj to allow us to pass information from our UI to our Csound code. In following code the max value of the SeekBar is set to 1 and the minium value is set to 0. To reserve this data in your Csound code use the opcode chnget and button1 and seekBar1. The button and SeekBar are added each time before every time CsoundObj is set playing to ensure they are in the variable cacheable for the object

```java
package com.example.first_csound_app;

import java.io.File;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.support.v4.app.NavUtils;
//THIS IS A COMMENT!// The following are the packages you need to import///
import android.widget.Button;
import android.widget.SeekBar;
import com.example.first_csound_app.R;
import com.csounds.CsoundObj;
import com.example.first_csound_app.BaseCsoundActivity;
////////////////////////////////////////////////////////////////////////////

public class MainActivity extends BaseCsoundActivity {

    private CsoundObj csoundObj;
    Button startCsound, stopCsound, button1;
    SeekBar seekBar1;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        csoundObj = new CsoundObj();
        startCsound = (Button) findViewById(R.id.StartCsound);
        stopCsound = (Button) findViewById(R.id.StopCsound);
        button1 = (Button) findViewById(R.id.Button1);;
        seekBar1 = (SeekBar) findViewById(R.id.SeekBar1);

        startCsound.setOnClickListener(new View.OnClickListener() {

            public void onClick(View v) {

                csoundObj.addSlider(seekBar1, "seekBar1", 0, 1);
                csoundObj.addButton(button1, "button1");
                csoundObj.startCsound(createTempFile(getResourceFileAsString(R.raw.button_test)));

            }
       });

        stopCsound.setOnClickListener(new View.OnClickListener() {

            public void onClick(View v) {

                csoundObj.stopCsound();


            }
        });

    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }
}
```

Finally, to get your Csound code into the project, create a *raw* folder in your project in the *res* folder. This can be done by right clicking your

res folder in your package explorer in Eclipse, from the drop down menu set *new* followed by *folder* then name the folder *raw*.



Next go to the containing folder of your project on your computer and put your Csound .csd file in the raw folder. Now when you return to Eclipse and run your project as an *Android Application* by clicking the play button in Eclipse, Your first Csound app should be running with your Csound code, congratulations!

9 Csound App 2.0

This app demonstrates the use of the Csound library in the context of a larger application with multiple Activities, Views and the ability to load in different .csd files dynamically. It is designed for stand alone use for those that do not want to get their hands dirty with Android programming but still want access to the platform, and for those who want an app with lots of functionality that they can use in their own applications.

Hence the in order to get up and running with Csound on Android, all you need to do is use chnget in your .csd to get access to the UI values.

SeekBars are set from 0 to 1 consequently some scaling may be necessary in your .csd. The Variable names to be used are:

- Performance1
  - seekBar1

  Performance 2
  - seekBar2

  Mixer
  - seekBar3
  - seekBar4
  - seekBar5
  - seekBar6
  - seekBar7
  - seekBar8
- Performance1
  - B1
  - B2
  - B3
  - B4
  - B5
  - B6
  - B7
  - B8
  - B9
  - B10
  - B11
  - B12
  - B13
  - B14
  - B15

- B16
- B17
- B18
- B19

Performance 2

- B20
- B21
- B22
- B23
- B24

Listing 8: Exampe of code to be used in .csd to connect Seekbars and Buttons

```
kVolume chnget "seekBar1"
kOnOff  chnget "B1"
```

To use the xy pad in tab Performance2 use the following

Listing 9: Exampe of code to be used in .csd to connect xy pad

```
S_xName sprintf "touch.%d.x", i_instanceNum
S_yName sprintf "touch.%d.y", i_instanceNum

kx chnget S_xName
ky chnget S_yName
```

For those who want to customise the app and understand its workings the following, along with the comments in the apps code should help. This application has been designed to allow user changes to the source code and was built to demonstrate the use of a singleton design process to preserve a CsoundObj for the whole Android lifecycle of an application. There are several situations that have to be taken into consideration.

1. Adding the UI controls to the CsoundObj when Android Activities are created and destroyed by the Android OS to recover RAM.

2. The selecting of new .csd files to be loaded into the application, and creating a new CsoundObj object for this event.

3. The consistency between different activities and views to allow users to change between views during a performance.

Many of these features are achieved by extending Application. It is used to store the CsoundObj, the current file path, the current features of each application and view. it is also used to give all activities a set of methods which will always have to be called and which will never be destroyed by the Android life cycle, thus ensuring their constant availability It is important to note that throughout this app BaseCsoundActivity.java is extended instead of Activity (As BaseCsoundActivity.java extends Acitivity). BaseCsoundActivity.java is used to take advantage of it methods such as:

- protected String getResourceFileAsString(int resId)
- protected File createTempFile(String csd)

Hence this may be a design choice worth remembering when coding your own apps.

10 Conclusion

We hope that this document has helped you to become familiar with the design and usage of the Csound for Android API. The example Java and Csound CSD code should hopefully give you a good starting point for your own musical projects, and we encourage you to take these examples and run with it. We look forward to hearing your questions and feedback on this API, and most of all, look forward to seeing what you will create with all of this. Best of luck and enjoy!