# Lab 4: Sending Data to Android from Pure Data

In this weeks lab we will extend our android app further. So far we have learned how to create a GUI for our app and how to send data from this GUI to our pd patch. Today we are going to learn how to send data from our patch to the GUI. This will allow us to display information in the GUI about the current state of the patch.

## Setup

Download the example from Github: <https://github.com/Hydroxate/Lab04_Template>

Rename the containing folder to firstname\_surname\_lab04.

(This step is important as otherwise another student may accidentally delete your project. )

Open the provided template in android studio.

## Edit The PD Patch

Lets create a simple counter in the PD patch.

1. Locate the raw folder in Android Studio.
2. Right click and locate the counter.zip in finder.
3. Unzip the archive.
4. Delete the archive.
5. Open the counter.pd file in pure data.
6. Edit the patch. Make the following changes:

Create a [metro] object with a default time of 1000ms.

Create a [float] object .

Create a [+ 1] object.

Connect the output of the [float] to the left inlet of the [+ 1] object.

Connect the output of the [+ 1] to the right inlet of the [float] object.

Create a [mod 10] object.

Connect the output of the float to the [mod 10] object.

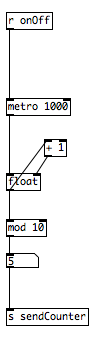
Create a Number.

Connect the output of the [mod 10] object to the Number.

Connect the output of the Number to the inlet of the provided [send sendCounter] object.

Test your patch in pd using the toggle provided. You should see the value from the float is increasing by 1 every second and resets to 0 after 9.

Your patch should be similar to the following figure.



### Include this new patch in the project

This is the same process as last week. The patch must be zipped to be included in the android app.

1. Save the patch
2. Close the patch
3. Right click the synth patch, and click “Compress counter.pd”.
4. This will create a zip file called “counter.pd.zip”. Rename it “counter.zip”.
5. Delete counter.pd

## Creating a GUI in Android Studio and Connecting it to a PD Patch.

We now want to create a widget in the GUI that will display the values created in our counter pd patch. How can we create a GUI that receives the values that go from 0 to 9 and displays them?

Well, we need to create a textView GUI widget in our app that will communicate with the [s sendCounter ] object in our PD patch. This is a three step process

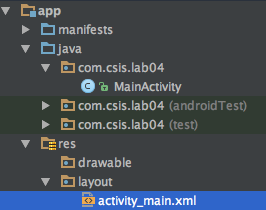
1. First we place the widget graphically on the screen of the app and set its properties such as location, color and most importantly, we give it a unique id.
2. Then we create an instance of the widget type (e.g. textView) in our code with a unique name and we link this to the graphical object we made in step 1.
3. Then we add the listener**.** We specify the [send] object we want to listen to. We specify a receiver to handle the received data from that [send] object. We subscribe to that [send] object to receive any future data. Then we add the functionality we want with the received data in the receiver.

### Add a TextView

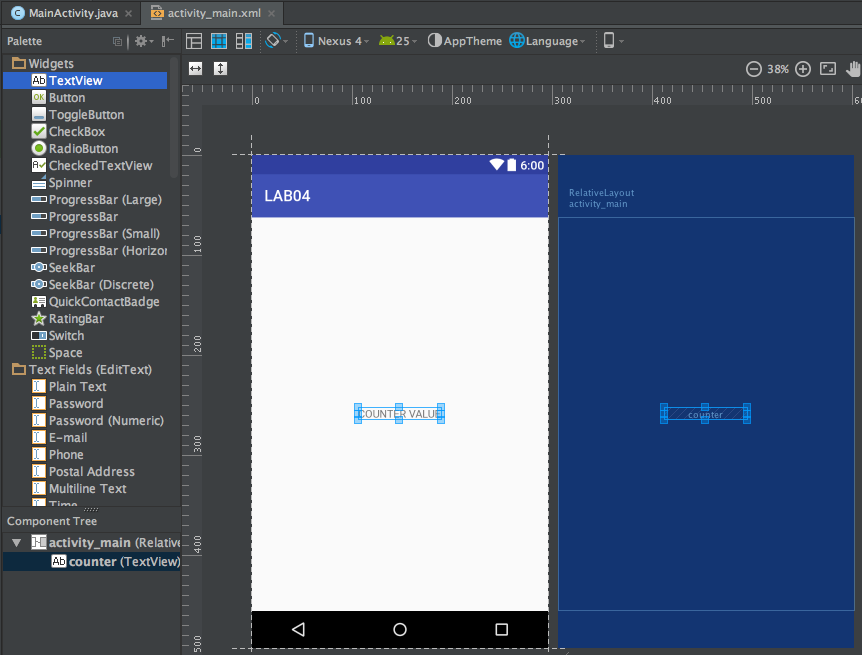
Lets follow the two step process and add a textView widget to our apps GUI.

### Step 1: Add the widget graphically

1. Click on activity\_main.xml in the project pane, it is located under res/layout.



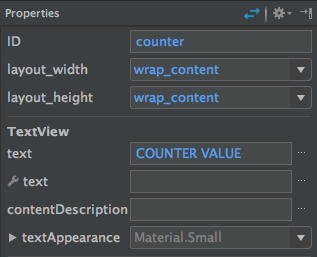
Drag and drop a TextView from the Widget Palette to create your GUI.



Click on the TextView widget that is now in place. Its properties should now be shown in the properties tab to the right hand side.

Now we need to set the properties for this new textView.

1. Go to the properties tab.



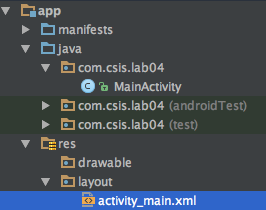
1. Rename the TextView’s ID to counter.
2. Rename the TextViews’s text to COUNTER VALUE

We have completed step 1 of our 3 step process for adding a widget to the gui.

### Step 2: Create an instance in code

At this stage our widget only exists as a graphical object, essentially a picture on the canvas. We have finished step 1 of our 2 step process. On to step two, “create an instance of the widget type in code with a unique name and link this to the graphical object we just made.”

1. Double click the MainActivity.java file in the project pane. This is located under the java folder.



Lets declare our Textview. Go to line 39 and type the below:



Now let’s create the instance of the code at line 46 in the OnCreate() method. It will look like the below:



When this line of code is executed it will create an instance of the TextView family called “myCounter” that is connected with the textView that has an ID of “counter”.

We have now completed step 2 of our three step process. We have created a TextView instance with a unique name (myCounter) and linked it to the graphical TextView object that we placed on the canvas in step 1.

Now we need to create our pure data listener!

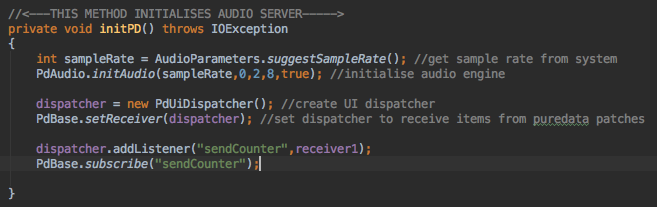
### Step 3: Add the pure data listener and receiver.

We want our app to receive data from our [send] objects in pure data. To do this we must add a listener first: code that listens for data coming from a [send] object. This needs to be added in the initPD() code which is at line 101.

1) Go to line 101 and type the below:

dispatcher.addListener("sendCounter",receiver1);  
PdBase.*subscribe*("sendCounter");

It should look like the following:



The line

dispatcher.addListener(“sendCounter”,receiver1)

creates a listener that will respond when something is sent from a [send sendCounter] object in pd. It sets up the listener to listen to the [send] object called “sendCounter” and if something should arrive send it to a receiver called receiver1. The receiver handles received data from patches. We will create the receiver in the next step.

The line PdBase.subscribe(“sendCounter”) makes the app continuously listen for data from the [send] object called “sendCounter”.

**To summarise, we add a listener. We specify the [send] object we want to listen to. We specify the receiver to handle the received data from that [send] object. We subscribe to that [send] object to receive any future data.**

### Create the receiver’s functionality

Now that we have added a listener, we must create the functionality of the receiver that processes the data.

1) Go to line 101 and type the following:

private PdReceiver receiver1 = new PdReceiver() {  
  
 private void pdPost(final String msg) {  
 Log.*e*("RECEIVED:", msg);  
  
 new Handler().post(new Runnable() {  
 @Override  
 public void run() {  
 }  
 });  
 }  
  
 @Override  
 public void print(String s) {  
 Log.*i*("PRINT",s);  
 Toast.*makeText*(getBaseContext(),s,Toast.*LENGTH\_LONG*);  
 }  
  
 @Override  
 public void receiveBang(String source)  
 {  
 pdPost("bang");  
 }  
  
 @Override  
 public void receiveFloat(String source, float x) {  
 pdPost("float: " + x);  
 if(source.equals("sendCounter")) {  
 myCounter.setText(String.*valueOf*(x));  
 }  
 }  
  
 @Override  
 public void receiveList(String source, Object... args) {  
 pdPost("list: " + Arrays.*toString*(args));  
  
 }  
  
 @Override  
 public void receiveMessage(String source, String symbol, Object... args) {  
 pdPost("message: " + Arrays.*toString*(args));  
 }  
  
 @Override  
 public void receiveSymbol(String source, String symbol) {  
 pdPost("symbol: " + symbol);  
 }  
};

It will look like the following:



There’s quite a bit of code here that whilst being necessary isnt really what we are interested in today. The functionality we are most interested in is the receiveFloat() method. Let’s take a look.



The receiveFloat method is called if the data that is sent from the [send] object is a float (a number essentially).

The receiveFloat method has two variables we can use: The String (called source) and the Float (called x). source will contain the name of the [send] object that the receiver got data from. x will contain the value that was sent. Read that a few more times. Its important.

The line beginning with “if” checks to see if the “source” matches the name “sendCounter”.

If it does then the next line

myCounter.setText(String.valueOf(x));

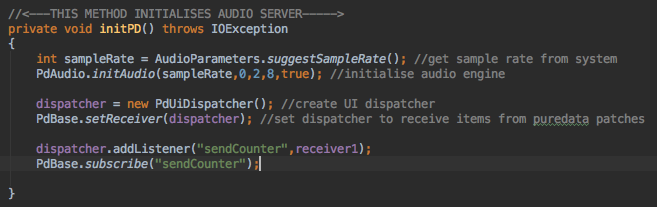
will execute. This line sets the text in the TextView widget called myCounter that we created earlier to the value of the received float. Essentially this is the line that will update the text in the textView widget to display the counter’s value!

1. Run your patch and see if there are errors.

You should see the text in the center updated from 0.0 to 9.0.

## Listening to more [send] object.

Back when we declared the listener in initPD() we specified the name of the [send] object and the receiver that would handle the incoming data.



To listen to other [send] objects we just add a listener and subscribe to that [send] object.

For example, listening to a [send] object in our patch called [s sendFrequency]:

Dispatcher.addListener(“sendCounter”,receiver1);

pdBase.subscribe(“sendCounter”);

Dispatcher.addListener(“sendFrequency”,receiver1);

pdBase.subscribe(“sendFrequency”);

The receiver we created earlier will listen for this other [send] object now!

Inside the receiver, we need to specify what happens with the data from [s sendFrequency]:

In the receiveFloat(String source, float x) method we can add the following code:

if(source.equals(”sendFrequency”))

{

//Functionality goes here.

}

This checks to see what is the source send object, if it equals “sendFrequency” then it will execute the code in the {}.

Your task is:

1. Edit your patch to send random numbers between 1 and 400 to a [send] object called sendFrequency.
2. Add a textView to your GUI with ID frequency.
3. Declare the textView in the mainActivity.java with name myFrequency.
4. Update the text of myFrequency to have the value of the received float from [s sendFrequency].

### Going Onwards

The homework for this lab will be sent out shortly. You’ll be required to add TextViews that will take data from [send] objects in a patch.

Use the rest of the lab to practice adding more textView boxes and updating them with daa from your patch. Can you come up with some useful examples? Maybe our patch has two states that it automatically switches between, say noisy and sine wavey and you want to pass that info to the GUI so you can tell hat state it is in.