Developer Documentation ComFrame

David Krebs, TU Munich David Zhang, TU Munich

October 15, 2014

Contents

1 What is ComFrame?

First of all some sentences about the purpose of ComFrame. It is an Open Source framework that allows the communication between two or more Android devices by sending and receiving sound.

2 How to get ComFrame?

Because ComFrame is Open Source getting it is quite easy: Just download the source from the website, or clone our git repository. We use the new Android Studio (beta as we're writing this) and a Gradle Build Process, that means starting developing with it is as easy as opening it in your prefered IDE, if there occur any differences at this point just search the internet for a solution for your specific IDE.

Once the import of the project is done, you will three different modules:

- **library** this one is the actual framework. It contains all classes you need to use ComFrame. To use it in your own app, just add this module as dependency to your own module (assuming you use gradle, too, this is quite easy)
- **example** in the example module there's a very basic example to show how to use ComFrame. It basically only allows you to send and receive text messages between two devices via sound.
- **TicTacToe** now here we go when you want to see a bigger example of the possibilities of ComFrame. It is a TicTacToe app you can play with one device against another.

3 How to use ComFrame?

The following section will be about how to use ComFrame in your own app. We assume you already have integrated the framework in your projekt, otherwise take a look at section 2. If there's anything unclear, it may always help to take a look at the example applications that were shipped with ComFrame.

First of all your application has to decide, if it will be the receiver or the sender. In the example apps we did this by giving the user the possibility to decide. Based on this decision take a look at the following two chapters that show how you can integrate either the Receiver or the Sender or maybe both (as done in the example TicTacToe app) into your application.

Important note: Because the framework will, as you may guess, emit and receive sound, you have to declare the following permission in your AndroidManifest.xml file:

```
<uses-permission android:name="android.permission.
    RECORD_AUDIO"></uses-permission>
```

3.1 ComFrame Sender

Using the ComFrame Sender is basically very easy, you just have to create a new instance (it makes sense to do this step when your application is starting). The returned instance should be saved to a field to send with it later on.

The whole process is shown here at one place, if you want to have further information about them just take a look at the explanations after this code block.

```
sender = new ComFrameSender();
// maybe set some options here
sender.prepare();
sender.send(data);
```

Further explanations about the single steps:

```
sender = new ComFrameSender();
```

After the instantiating you're able to set some options at the Sender, for example the frequency or debug flags. Because this part is mainly the same as on the Receiver side, continue to section 4 for more information. Note also that setting additionally options is not necessary, the default options are just fine.

When the new instance is created and you may have set additional options, we recommend to call prepare(). Although it isn't necessary, because it will be called when you call send(data) the first time, it ensures that send(data) starts immediately with sending because everything is already prepared.

```
// set option (not necessary)
sender.debug = true;
sender.prepare();
```

After this steps, you just can start sending data. This is achieved as following, replacing the data with your own byte array.

```
byte[] data = {0, 1, 2, 3, 4};
sender.send(data);
```

Note at this point that the current version of ComFrame only supports sending not more than 255 bytes at once, counting the bytes after an eventual hamming decoding is done! You will get an exception if you try to send byte arrays larger than that restriction.

Another important thing you should consider is that the send(data) method returns as soon as all data were sent, it blocks until then. That has the advantage that the application can easy determine when sending is complete, but one disadvantage may be that you have to call it in a seperate thread if you want to compute something different in the meantime.

Once your application will not use the ComFrameSender any more, call

```
sender.close();
```

to release the AudioTrack properly.

For further questions take a look at the example applications.

3.2 ComFrame Receiver

The ComFrameReceiver can be used similiar to the the Sender. Again, the whole procedure in one block:

```
receiver = new ComFrameReceiver();
// set at least one of the listener
receiver.setMsgListener(msgListener);
receiver.setStreamListener(streamListener);
receiver.setBitListener(bitListener);
// maybe set some other options
receiver.debug = true;
receiver.prepare()

receiver.startListening();
```

Further explanations about the single steps:

First of all start by creating a new ComFrameReceiver instance and of course save it somewhere:

```
receiver = new ComFrameReceiver();
```

Now after you have your new instance one of the most important steps follows: You should pass at least one listener to the ComFrameReceiver such that you will be informed about new data received. The listener interfaces can be found in the class ComFrame, that means you have to implement them in one of your classes The three different interfaces you have to choose from are:

- **MsgListener:** this listener always stores the received bytes and passes them to you once the whole message from the Sender was received.

```
class A implements ComFrame.MsgListener {
    public void onMessageReceived(byte[] msg){
        // process the message here
    }
}
```

- **StreamListener:** use this interface if you want to get notified for every single byte received from the Sender. This interface can be useful if you want to display some kind of progress bar. Once the whole message was received, ready() will be called.

```
class A implements ComFrame.StreamListener {
    public void onByteReceived(byte b){
        // process the byte here
    }

    public void ready(){
        // received the whole message
    }
}
```

- **BitListener:** this listener informes you about every single bit received. CAUTION: it gives you the raw bits before hamming decoding is applied. This interface is only if you're interested in further debugging.

```
class A implements ComFrame.BitListener {
    public void onBitReceived(boolean bit){
        // process the bit here
    }
}
```

Once you have implemented at least one of the interfaces, you can set the corresponding listener at the ComFrameReceiver:

```
receiver.setMsgListener(msgListener);
receiver.setStreamListener(streamListener);
receiver.setBitListener(bitListener);
```

At this point you can also set some additional options such as the debugging flag. Because this part is the same for Sender and Receiver, it is described in section 4 , take a look at it if you're interested, but we want to point out that the default settings are tested the most and should therefor work the best.

Now after setting all listener and options you should call the prepare() method. Although it's not necessary, we recommend doing it, then the Receiver won't have to do the preparation later when it starts listening for data.

```
receiver.prepare();
```

All you have to do then is to start listening, and you will be informed about new bytes through the listener you should have set above.

```
receiver.startListing();
```

Note that you can stop and start listening for data at any time by just calling the startListing() and stopListing() method:

```
receiver.stopListing();
```

4 Setting Additional Options

You can set some options additionally to the once we described up to now. They will be described only for the receiver, but can be (and some should be!) set the same way for the sender.

- **debug** switch: set this flag to receive further debug information, but you should switch it off before deploying, of course.

```
receiver.debug = true;
```

- **verbose** switch: for even more information set this flag, but attention! it may spam your logcat and is mainly for developing of ComFrame.

```
receiver.verbose = true;
```

- **Hamming Code:** if you want your data to be decoded and encoded with an hamming code, you can use this setter (HammingCode is an enum):

```
receiver.setHammingCode(HammingCode code);
```

  Currently we support this hamming encodings:

  - HammingCode.HAMMING_7_4
  - HammingCode.HAMMING_15_11

  Important: set the same hamming code for the receiver as for the sender, of course!