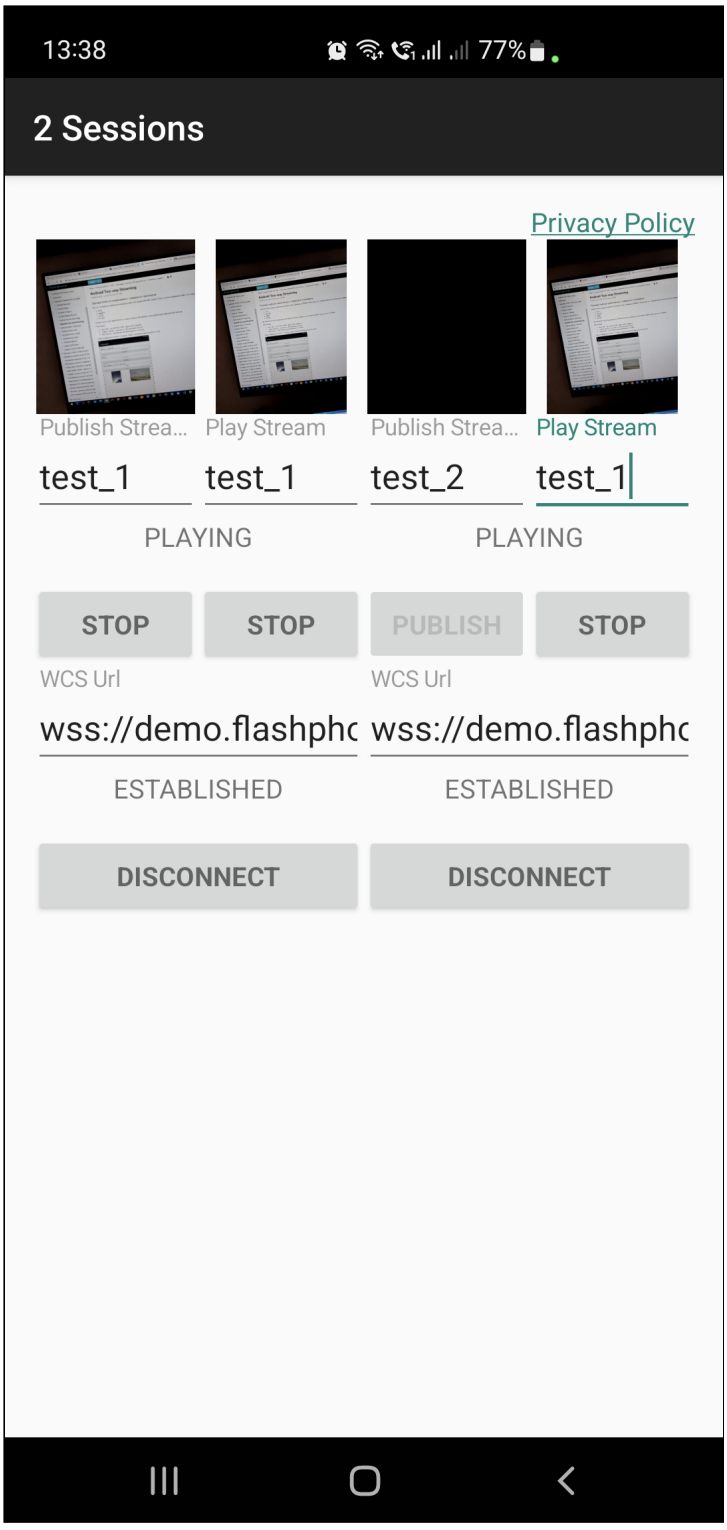
Android 2 Sessions

Android application example with two Websocket sessions

This application shows how to use two Websocket sessions simultaneously to connect to one or more WCS instances. Only one stream can be published in this case because Android does not allow to capture two cameras simultaneously or two streams from the same camera. However, two streams can be played simultaneously.

Screenshot below shows publishing and playback in two sessions with one WCS server



Analyzing the example code

To analyze the code take TwoSessionsActivity.java class from `2sessions` example available in build 1.1.0.57.

The implementation itself is like two Two Way Streaming examples in one application, so it's enough to explain one session code

1. API initialization

`Flashphoner.init()` code

API should be initialized once per application regardless of sessions number

```
Flashphoner.init(this);
```

2. Session creation

`Flashphoner.createSession()` code

`SessionOptions` object is passed to `createSession` method with the following parameters

- URL of WCS server
- `SurfaceViewRenderer local1Renderer`, which will be used to display video from the camera
- `SurfaceViewRenderer remote1Renderer`, which will be used to play video stream

```
SessionOptions sessionOptions = new
SessionOptions(mWcsUrl1View.getText().toString());
sessionOptions.setRemoteRenderer(remote1Render);
sessionOptions.setLocalRenderer(local1Render);

/**
 * Session for connection to WCS server is created with method
createSession().
 */
session1 = Flashphoner.createSession(sessionOptions);
```

3. Connecting to the server

`Session.connect()` code

```
session1.connect(new Connection());
```

4. Receiving the event confirming successful connection

`session.onConnected()` code

Access permissions to a camera and microphone are requested on this event

```
@Override
public void onConnected(final Connection connection) {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
```

```
            mConnect1Button.setText(R.string.action_disconnect);
            mConnect1Button.setTag(R.string.action_disconnect);
            mConnect1Button.setEnabled(true);
            mConnect1Status.setText(connection.getStatus());
            mPlay1Button.setEnabled(true);
            if (mPublish2Button.getTag() == null ||
Integer.valueOf(R.string.action_publish).equals(mPublish2Button.getTag())) {
                mPublish1Button.setEnabled(true);
            }

            ActivityCompat.requestPermissions(TwoSessionsActivity.this,
                    new String[]{Manifest.permission.RECORD_AUDIO,
Manifest.permission.CAMERA},
                    PUBLISH_REQUEST_CODE);
        }
    });
}
```

5. Stream publishing by `Publish1Button` click

`Session.createStream()`, `Stream.publish()` code

```
mPublish1Button.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View view) {
        mPublish1Button.setEnabled(false);
        mPublish2Button.setEnabled(false);
        mConnect1Button.setEnabled(false);
        mConnect2Button.setEnabled(false);
        if (mPublish1Button.getTag() == null ||
Integer.valueOf(R.string.action_publish).equals(mPublish1Button.getTag())) {
            /**
             * The options for the stream to play are set.
             * The stream name is passed when StreamOptions object is
created.
             * SurfaceViewRenderer to be used to display the video stream is
set using method StreamOptions.setRenderer().
             */
            StreamOptions streamOptions = new
StreamOptions(mPublish1StreamView.getText().toString());
            streamOptions.setRenderer(local1Render);

            /**
             * Stream is created with method Session.createStream().
             */
            publish1Stream = session1.createStream(streamOptions);
            ...
            publish1Stream.publish();
        } else {
            ...
        }
    }
});
```

6. Stream playback by `Play1Button` click

`Session.createStream()`, `Stream.play()` code

```
mPlay1Button.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View view) {
        mPlay1Button.setEnabled(false);
        if (mPlay1Button.getTag() == null ||
Integer.valueOf(R.string.action_play).equals(mPlay1Button.getTag())) {
            /**
             * The options for the stream to play are set.
             * The stream name is passed when StreamOptions object is
created.
             * SurfaceViewRenderer to be used to display the video stream is
set using method StreamOptions.setRenderer().
             */
            StreamOptions streamOptions = new
StreamOptions(mPlay1StreamView.getText().toString());
            streamOptions.setRenderer(remote1Render);

            /**
             * Stream is created with method Session.createStream().
             */
            play1Stream = session1.createStream(streamOptions);
            ...
            play1Stream.play();
            ...
        } else {
            ...
        }
    }
});
```

7. Stream playback stopping

`Stream.stop()` code

```
mPlay1Button.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View view) {
        mPlay1Button.setEnabled(false);
        if (mPlay1Button.getTag() == null ||
Integer.valueOf(R.string.action_play).equals(mPlay1Button.getTag())) {
            ...
        } else {
            /**
             * Method Stream.stop() is called to stop playback of the stream.
             */
            play1Stream.stop();
            play1Stream = null;
        }
        ...
```

```
    }
});
```

8. Stream publishing stopping

`Stream.stop()` code

```
mPublish1Button.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View view) {
        mPublish1Button.setEnabled(false);
        mPublish2Button.setEnabled(false);
        mConnect1Button.setEnabled(false);
        mConnect2Button.setEnabled(false);
        if (mPublish1Button.getTag() == null ||
Integer.valueOf(R.string.action_publish).equals(mPublish1Button.getTag())) {
            ...
        } else {
            /**
             * Method Stream.stop() is called to stop playback of the stream.
             */
            publish1Stream.stop();
            publish1Stream = null;
        }
        ...
    }
});
```

9. Connection closing

`Session.disconnect()` code

```
mConnect1Button.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View view) {
        if (mConnect1Button.getTag() == null ||
Integer.valueOf(R.string.action_connect).equals(mConnect1Button.getTag())) {
            ...
        } else {
            mConnect1Button.setEnabled(false);

            /**
             * Connection to WCS server is closed with method
Session.disconnect().
             */
            session1.disconnect();
        }
        ...
    }
});
```

10. Receiving event confirming disconnection

`session.onDisconnection()` code

```
public void onDisconnection(final Connection connection) {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            mConnect1Button.setText(R.string.action_connect);
            mConnect1Button.setTag(R.string.action_connect);
            mConnect1Button.setEnabled(true);
            mPlay1Button.setText(R.string.action_play);
            mPlay1Button.setTag(R.string.action_play);
            mPlay1Button.setEnabled(false);
            mPublish1Button.setEnabled(false);
            mPublish1Button.setText(R.string.action_publish);
            mPublish1Button.setTag(R.string.action_publish);
            mConnect1Status.setText(connection.getStatus());
            mPlay1Status.setText("");
            if (mConnect2Button.getTag() == null ||
Integer.valueOf(R.string.action_disconnect).equals(mConnect2Button.getTag()))
{
                mPublish2Button.setEnabled(true);
            }
        }
    });
}
```