

Android Two-way Streaming

Example of Android application with player and streamer

This streamer can be used to publish WebRTC video stream and play any of the following types of streams on Web Call Server

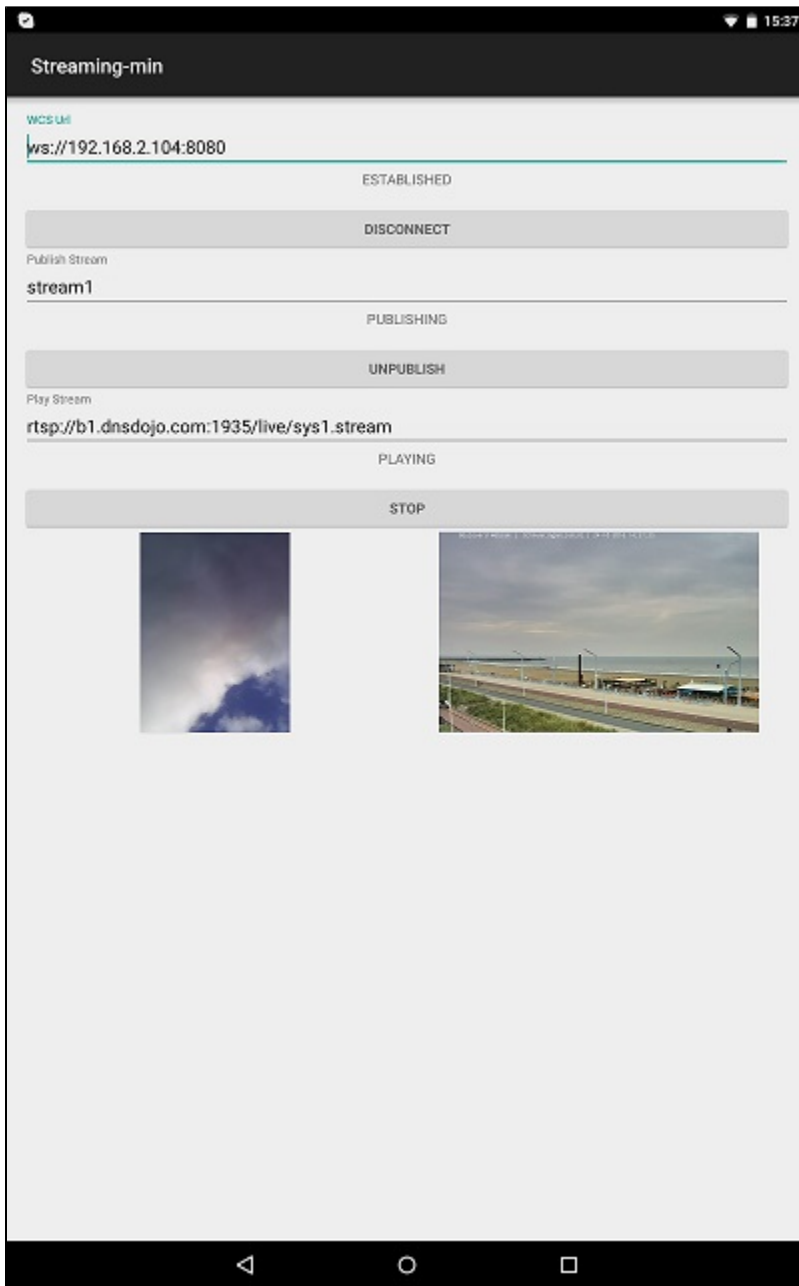
- RTSP
- WebRTC
- RTMP
- RTMFP

On the screenshot below the example is displayed when a stream is being published and another stream is being played.
Input fields

- 'WCS URL', where 192.168.2.104 is the address of the WCS server
- 'Publish Stream' - for the name of published stream
- 'Play Stream' - for the name of played stream

Two videos are played

- left - video from the camera
- right - the played video stream



Work with code of the example

To analyze the code, let's take class [StreamingMinActivity.java](#) of the streaming-min example, which can be downloaded with corresponding build [1.0.1.38](#).

1. Initialization of the API.

Flashphoner.init()[code](#)

For initialization, object Context is passed to the init() method.

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

2. Session creation.

Flashphoner.createSession()[code](#)

Object SessionOptions ([line 88](#)) with the following parameters is passed to createSession() method:

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

```
sessionOptions = new SessionOptions(mWcsUrlView.getText().toString());
sessionOptions.setLocalRenderer(localRender);
sessionOptions.setRemoteRenderer(remoteRender);

/**
 * Uncomment this code to use your own RTCCongfiguration. For example, you can use custom TURN server
 */
//List<PeerConnection.IceServer> iceServers = new ArrayList<>();
//iceServers.add(new PeerConnection.IceServer("turn:your.turn-server.com:443?transport=tcp","username","
passw0rd"));
//PeerConnection.RTCCongfiguration customConfig = new PeerConnection.RTCCongfiguration(iceServers);
//sessionOptions.setMediaOptions(customConfig);

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

3. Connection to the server.

Session.connect()[code](#)

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

4. Receiving the event confirming successful connection.

session.onConnected()[code](#)

```
@Override
public void onConnected(final Connection connection) {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            mConnectButton.setText(R.string.action_disconnect);
            mConnectButton.setTag(R.string.action_disconnect);
            mConnectButton.setEnabled(true);
            mConnectStatus.setText(connection.getStatus());
            mPublishButton.setEnabled(true);
            mPlayButton.setEnabled(true);
        }
    });
}
```

5. Permissions request to publish video stream on "Publish" button pressing

ActivityCompat.requestPermissions()[code](#)

```
mPublishButton.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View view) {
        if (mPublishButton.getTag() == null || Integer.valueOf(R.string.action_publish).equals(mPublishButton.
getTag())) {
            ActivityCompat.requestPermissions(StreamingMinActivity.this,
                new String[]{Manifest.permission.RECORD_AUDIO, Manifest.permission.CAMERA},
                PUBLISH_REQUEST_CODE);
            ...
        } else {
            ...
        }
    }
});
```

6. Video streaming when permissions are granted.

Session.createStream(), Stream.publish()[code](#)

```
case PUBLISH_REQUEST_CODE: {
    if (grantResults.length == 0 ||
        grantResults[0] != PackageManager.PERMISSION_GRANTED ||
        grantResults[1] != PackageManager.PERMISSION_GRANTED) {
        Log.i(TAG, "Permission has been denied by user");
    } else {
        mPublishButton.setEnabled(false);
        /**
         * The options for the stream to publish are set.
         * The stream name is passed when StreamOptions object is created.
         */
        StreamOptions streamOptions = new StreamOptions(mPublishStreamView.getText().toString());

        /**
         * Uncomment this code to use case WebRTC-as-RTMP. Stream will be republished to your rtmpUrl
         */
        //streamOptions.setRtmpUrl("rtmp://192.168.1.100:1935/live2");

        /**
         * Stream is created with method Session.createStream().
         */
        publishStream = session.createStream(streamOptions);
        ...
        /**
         * Method Stream.publish() is called to publish stream.
         */
        publishStream.publish();

        Log.i(TAG, "Permission has been granted by user");
    }
}
```

7. Stream playback on "Play" button pressing

Session.createStream(), Stream.play()[code](#)

```
mPlayButton.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View view) {
        mPlayButton.setEnabled(false);
        if (mPlayButton.getTag() == null || Integer.valueOf(R.string.action_play).equals(mPlayButton.getTag()))
        {
            /**
             * The options for the stream to play are set.
             * The stream name is passed when StreamOptions object is created.
             */
            StreamOptions streamOptions = new StreamOptions(mPlayStreamView.getText().toString());

            /**
             * Stream is created with method Session.createStream().
             */
            playStream = session.createStream(streamOptions);
            ...
            /**
             * Method Stream.play() is called to start playback of the stream.
             */
            playStream.play();
            ...
        } else {
            ...
        }
    }
});
```

8. Stream playback stop on "Stop" button pressing

Stream.stop()[code](#)

```
playStream.stop();  
playStream = null;
```

9. Streaming stop on "Unpublish" button pressing

Stream.stop()[code](#)

```
publishStream.stop();  
publishStream = null;
```

10. Session disconnection.

Session.disconnect()[code](#)

```
mConnectButton.setEnabled(false);  
  
/**  
 * Connection to WCS server is closed with method Session.disconnect().  
 */  
session.disconnect();
```

11. Receiving the event confirming successful disconnection.

session.onDisconnection()[code](#)

```
@Override  
public void onDisconnection(final Connection connection) {  
    runOnUiThread(new Runnable() {  
        @Override  
        public void run() {  
            mConnectButton.setText(R.string.action_connect);  
            mConnectButton.setTag(R.string.action_connect);  
            mConnectButton.setEnabled(true);  
            mPublishButton.setText(R.string.action_publish);  
            mPublishButton.setTag(R.string.action_publish);  
            mPublishButton.setEnabled(false);  
            mPlayButton.setText(R.string.action_play);  
            mPlayButton.setTag(R.string.action_play);  
            mPlayButton.setEnabled(false);  
            mConnectStatus.setText(connection.getStatus());  
            mPublishStatus.setText("");  
            mPlayStatus.setText("");  
        }  
    });  
}
```