

Android Streamer

Example of streamer for Android

This streamer can be used to publish WebRTC video stream on Web Call Server.

On the screenshot below the example is displayed when a stream is being published.

In the URL specified in the input field

- `192.168.2.104` is the address of the WCS server
- `test` is the stream name

Two videos are played

- left - video from the camera
- right - the published video stream as received from the server



Analyzing the example code

To analyze the code, let's take class [StreamerActivity.java](#) of the `streamer` example, which can be downloaded with corresponding build [1.0.1.38](#).

1. Initialization of the API

`Flashphoner.init()` [code](#)

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

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

2. Session creation

`Flashphoner.createSession()` [code](#)

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

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

```
SessionOptions sessionOptions = new SessionOptions(url);
sessionOptions.setLocalRenderer(localRender);
sessionOptions.setRemoteRenderer(remoteRender);

/**
 * 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() {
            mStartButton.setText(R.string.action_stop);
            mStartButton.setTag(R.string.action_stop);
            mStartButton.setEnabled(true);
            mStatusView.setText(connection.getStatus());

            /**
             * The options for the stream to publish are set.
             * The stream name is passed when StreamOptions object is
             * created.
             */
            StreamOptions streamOptions = new StreamOptions(streamName);
```

```

        /**
         * Stream is created with method Session.createStream().
         */
        publishStream = session.createStream(streamOptions);
        ...
    }
});
}

```

5. Video stream creation

`Session.createStream()`, `ActivityCompat.requestPermissions()` [code](#)

`StreamOptions` object with name of the stream is passed to the `createStream()` method.

```

StreamOptions streamOptions = new StreamOptions(streamName);

/**
 * Stream is created with method Session.createStream().
 */
publishStream = session.createStream(streamOptions);
...
ActivityCompat.requestPermissions(StreamerActivity.this,
    new String[]{Manifest.permission.RECORD_AUDIO,
        Manifest.permission.CAMERA},
    PUBLISH_REQUEST_CODE);

```

6. Video stream publishing

`Stream.publish()` [code](#)

```

case PUBLISH_REQUEST_CODE: {
    if (grantResults.length == 0 ||
        grantResults[0] != PackageManager.PERMISSION_GRANTED ||
        grantResults[1] != PackageManager.PERMISSION_GRANTED) {
        mStartButton.setEnabled(false);
        session.disconnect();
        Log.i(TAG, "Permission has been denied by user");
    } else {
        /**
         * Method Stream.publish() is called to publish stream.
         */
        publishStream.publish();
        Log.i(TAG, "Permission has been granted by user");
    }
}
}

```

7. Receiving the event confirming successful stream publishing

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

On receiving this event preview stream is created with `Session.createStream()` and `Stream.play()` is invoked to play it.

```
publishStream.on(new StreamStatusEvent() {
    @Override
    public void onStreamStatus(final Stream stream, final StreamStatus
streamStatus) {
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                if (StreamStatus.PUBLISHING.equals(streamStatus)) {

                    /**
                     * The options for the stream to play are set.
                     * The stream name is passed when StreamOptions object
is created.
                     */
                    StreamOptions streamOptions = new
StreamOptions(streamName);

                    /**
                     * 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 {
                    Log.e(TAG, "Can not publish stream " + stream.getName() +
" " + streamStatus);
                }
                mStatusView.setText(streamStatus.toString());
            }
        });
    }
});
```

8. Session disconnection

`Session.disconnect()` code

```
mStartButton.setEnabled(false);

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

9. Receiving the event confirming successful disconnection

`Session.onDisconnection()` [code](#)

```
@Override
public void onDisconnection(final Connection connection) {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            mStartButton.setText(R.string.action_start);
            mStartButton.setTag(R.string.action_start);
            mStartButton.setEnabled(true);
            mStatusView.setText(connection.getStatus());
        }
    });
}
```