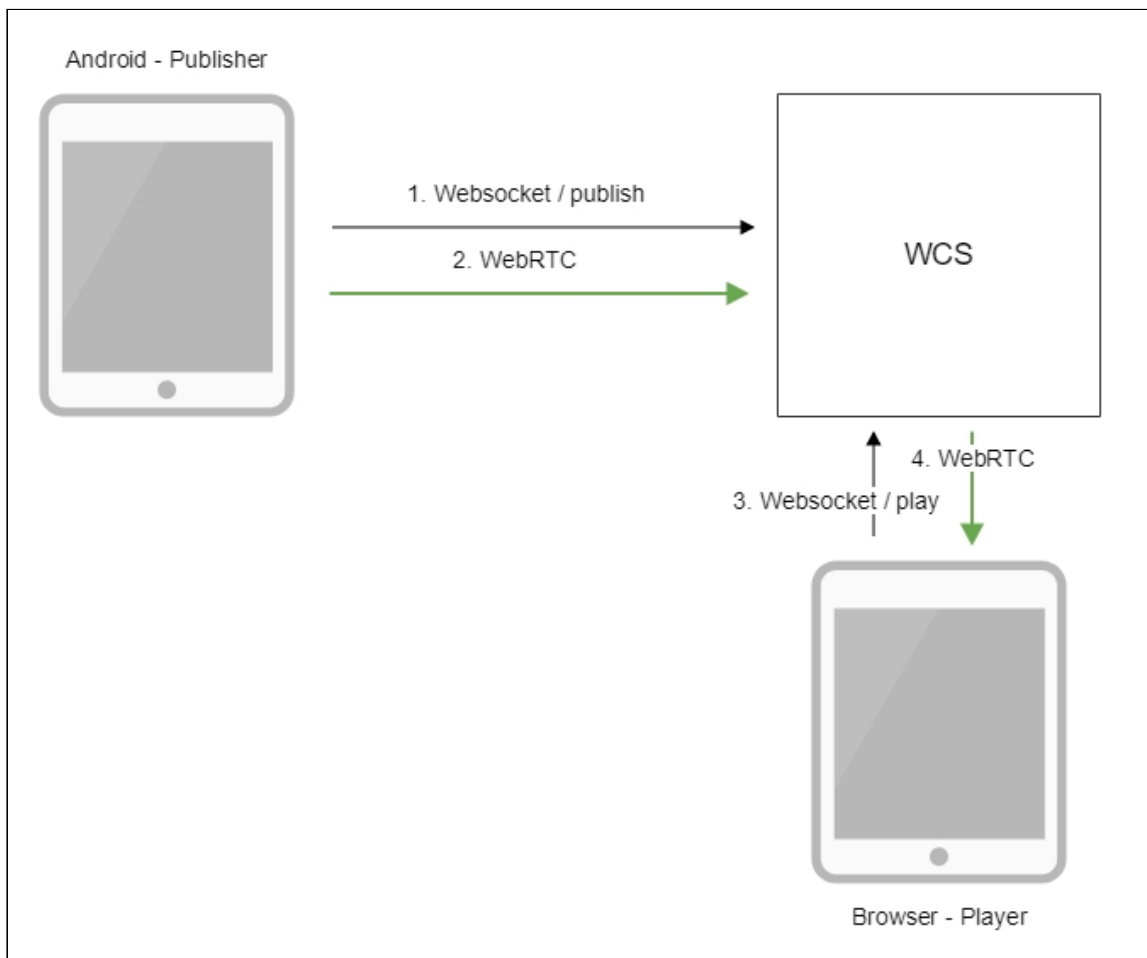


From an Android mobile app via WebRTC

Overview

WCS provides SDK to develop client applications for the Android platform

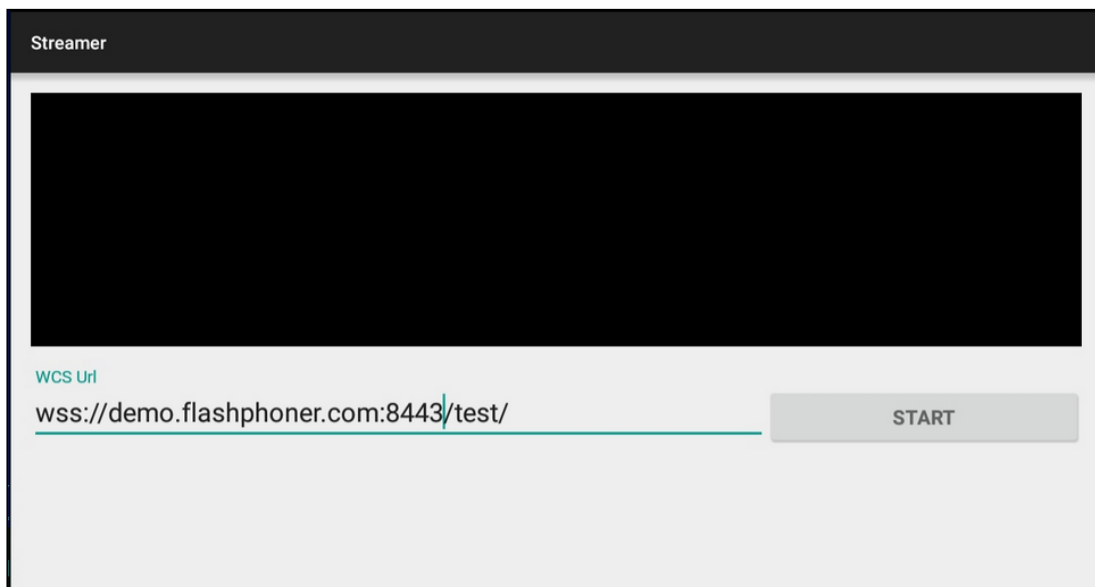
Operation flowchart



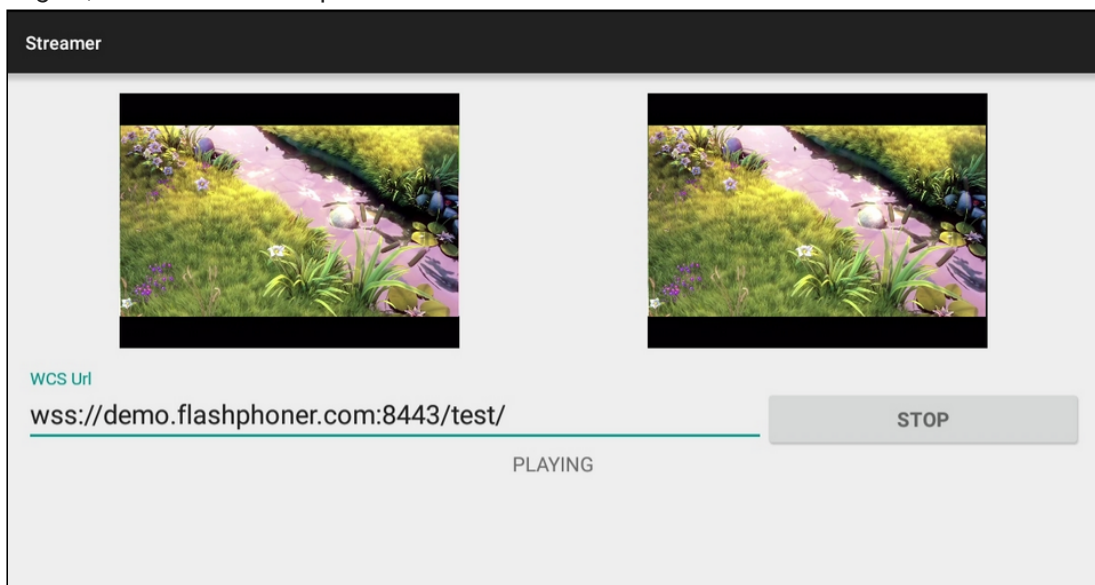
1. The Android device connects to the server via the Websocket protocol and sends the `publishStream` command.
2. The Android device captures the microphone and the camera and sends the WebRTC stream to the server.
3. The browser establishes a connection via WebSocket and sends the `playStream` command.
4. The browser receives the WebRTC stream and plays that stream on the page.

Quick manual on testing

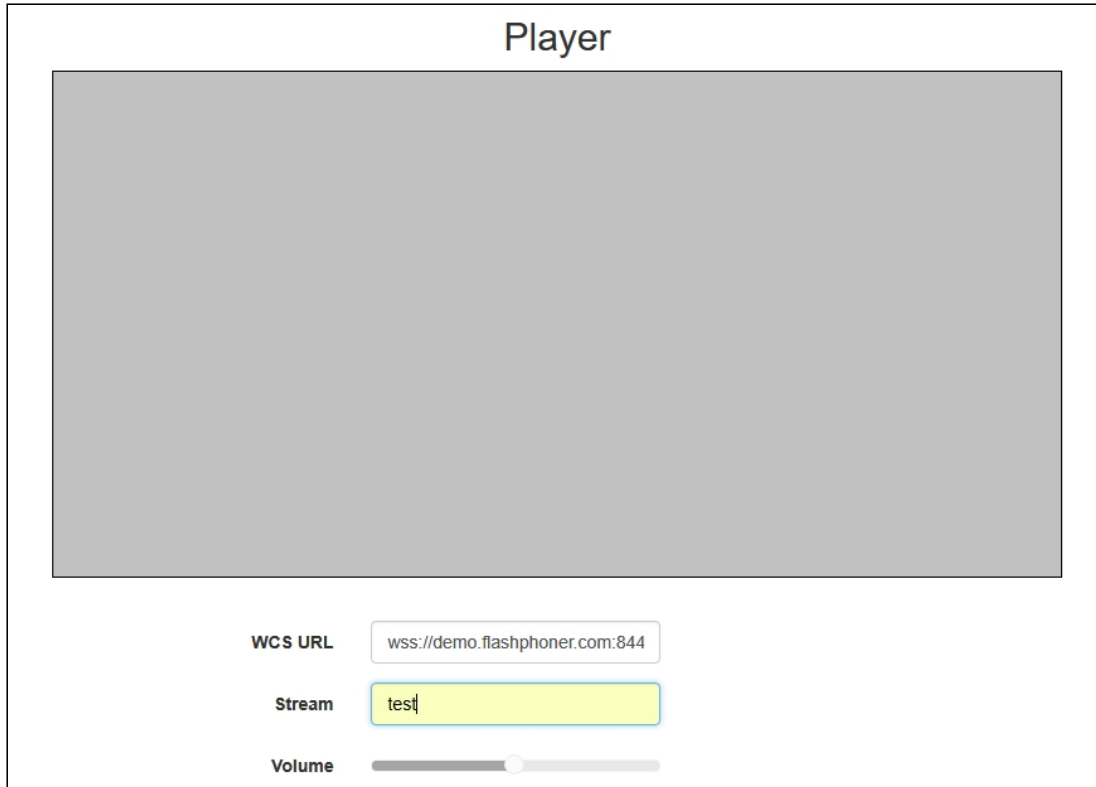
1. For this test we use:
2. the demo server at `demo.flashphoner.com`;
3. the Streamer mobile app ([Google Play](#));
4. the [Player](#) web application to display the captured stream
5. Install the Streamer mobile app on the Android device from [Google Play](#). Start the app on the device and enter the URL as `wss://demo.flashphoner.com:8443/test/`, where `demo.flashphoner.com` is the address of the WCS server, `test` is the identifier of the stream:



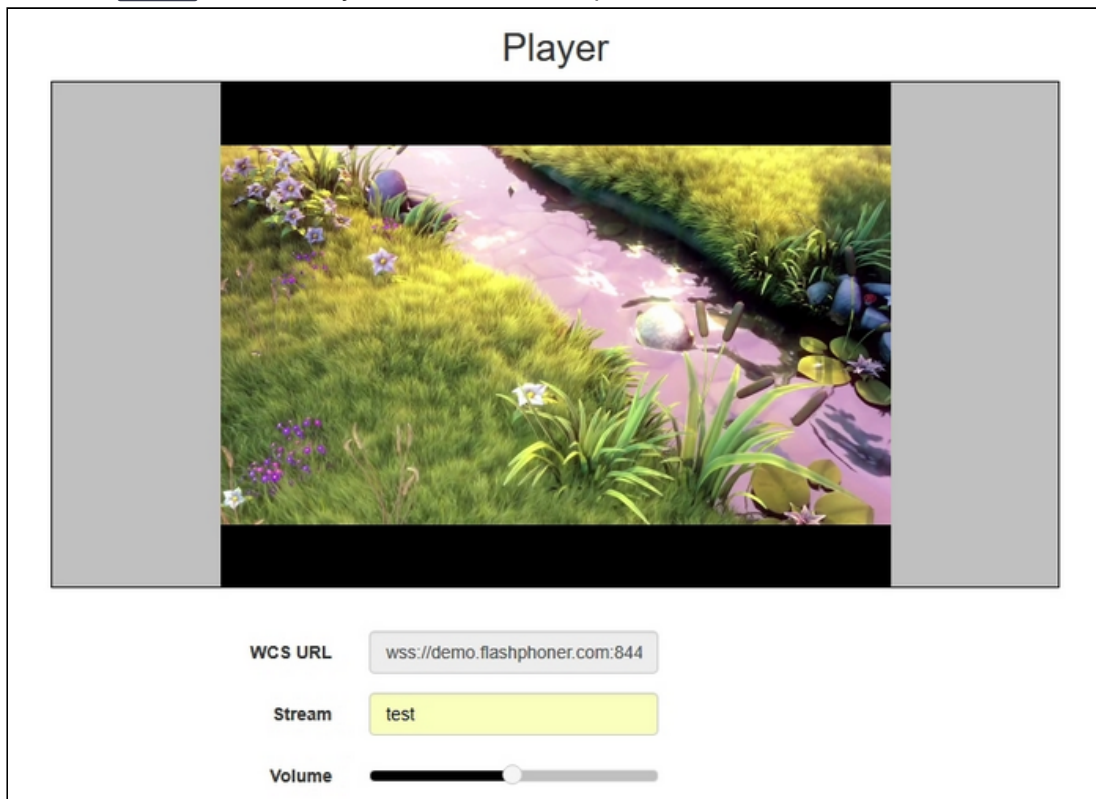
6. Click the **Start** button. Capturing of a video stream from the front camera of the device begins, and the stream is published to the server:



7. Open the Player web application, specify the stream identifier `test` in the `Stream` field:



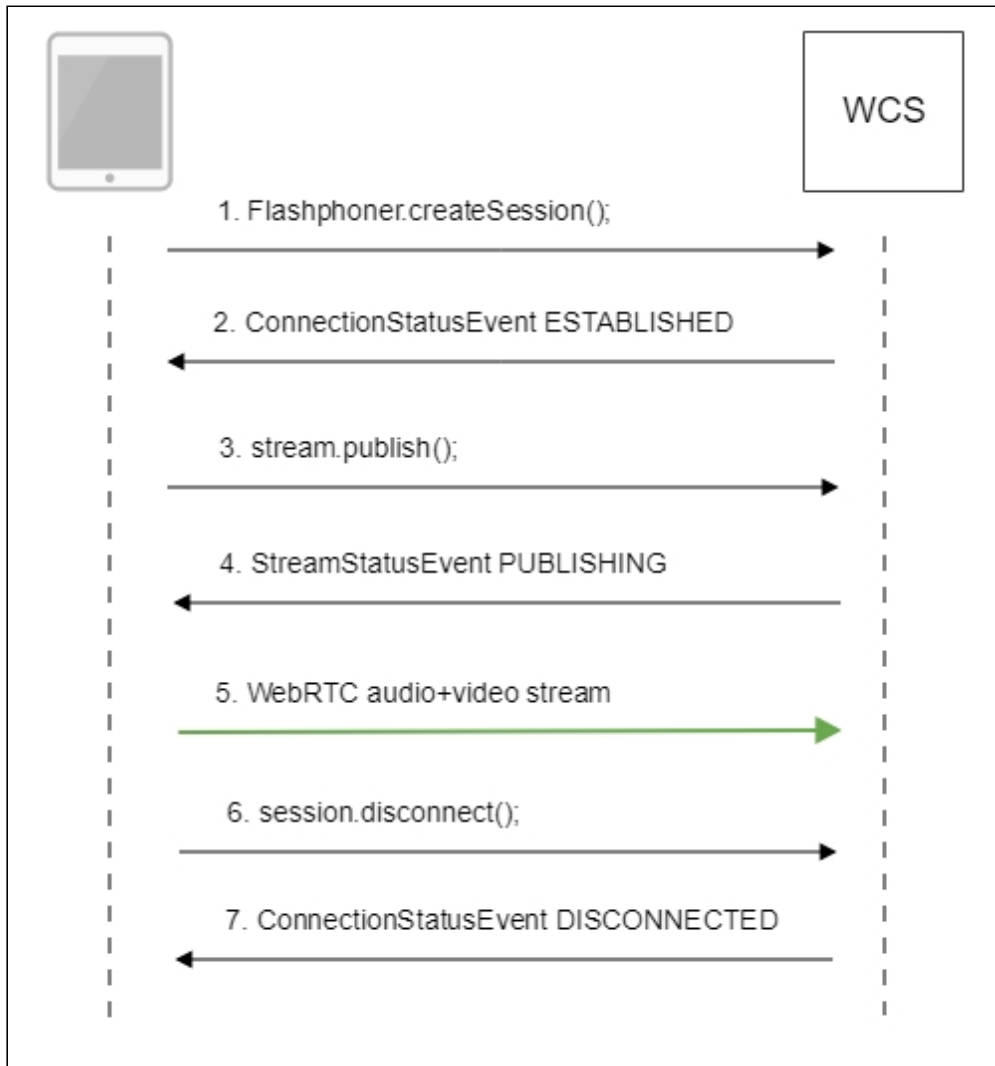
8. Click the `Start` button. Playback of the stream published from the mobile device starts:



Call flow

Below is the call flow when using the Streamer example

[StreamerActivity.java](#)



1. Establishing connection to the server

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

```
String url;
final String streamName;
try {
    URI u = new URI(mWcsUrlView.getText().toString());
    url = u.getScheme() + "://" + u.getHost() + ":" + u.getPort();
    streamName = u.getPath().replaceAll("/", "");
} catch (URISyntaxException e) {
    mStatusView.setText("Wrong uri");
    return;
}

/**
 * The options for connection session are set.
 * WCS server URL is passed when SessionOptions object is created.
 * SurfaceViewRenderer to be used to display video from the camera is set
```

```

with method SessionOptions.setLocalRenderer().
* SurfaceViewRenderer to be used to display preview stream video received
from the server is set with method SessionOptions.setRemoteRenderer().
*/
SessionOptions sessionOptions = new SessionOptions(url);
sessionOptions.setLocalRenderer(localRender);
sessionOptions.setRemoteRenderer(remoteRender);

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

```

2. Receiving from the server an event confirming successful connection

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

```

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);
            ...
        }
        ...
    });
    ...
}

```

3. Publishing the stream

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

```

@Override
public void onRequestPermissionsResult(int requestCode,
                                       @NonNull String permissions[],
                                       @NonNull int[] grantResults) {
    switch (requestCode) {
        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");
        }
    }
}
}
}
}

```

4. Receiving from the server an event confirming successful publishing of the stream

`Stream.onStreamStatus()`, `StreamStatus.PUBLISHING` [code](#)

```

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);
                    ...
                }
                ...
            }
        });
        ...
    }
});
...
});

```

5. Sending audio and video stream via WebRTC

6. Stopping publishing the stream

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

```

if (mStartButton.getTag() == null ||
Integer.valueOf(R.string.action_start).equals(mStartButton.getTag())) {
    ...
} else {
    mStartButton.setEnabled(false);

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

```

7. Receiving from the server and event confirming successful unpublishing of the stream

`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());
        }
    });
}

```