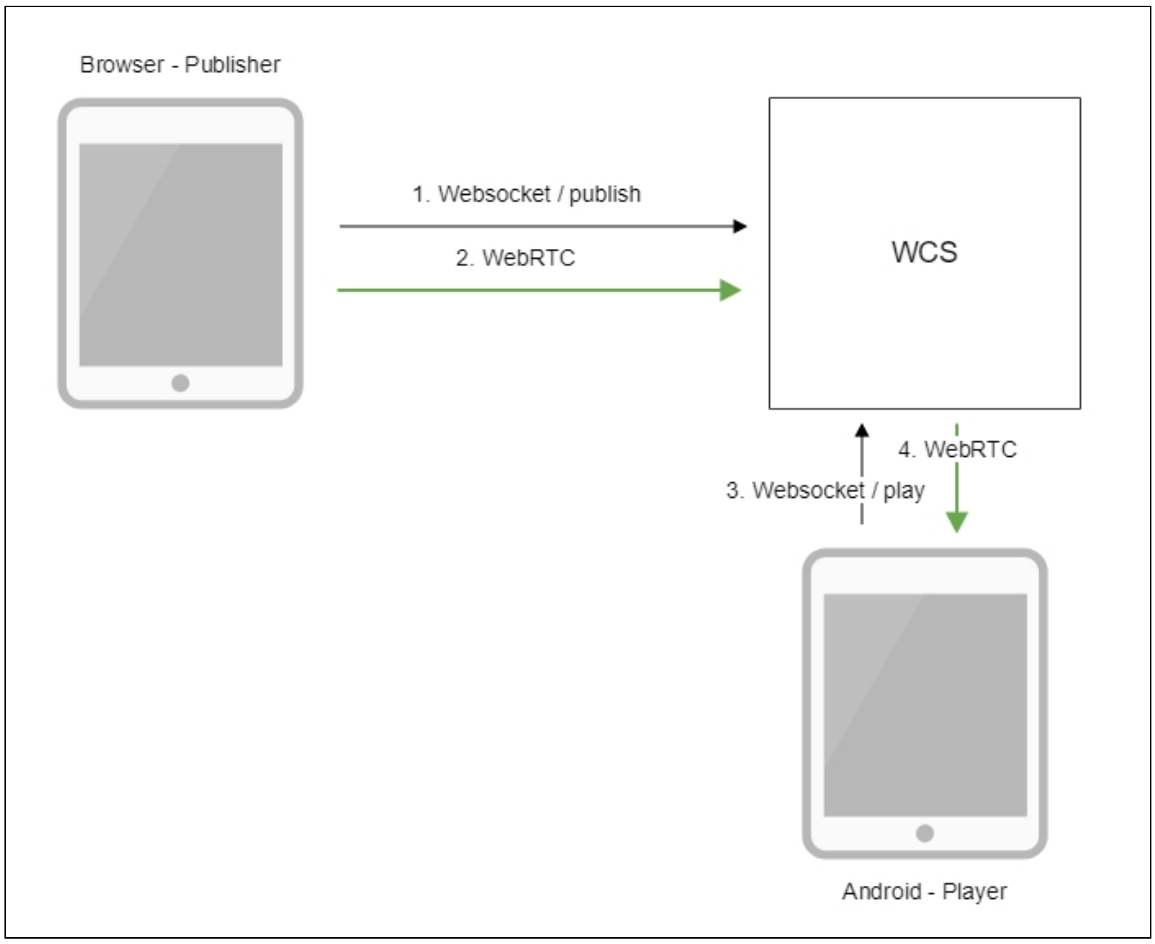
In an Android mobile application via WebRTC

Overview

WCS provides SDK to develop client applications for the Android platform

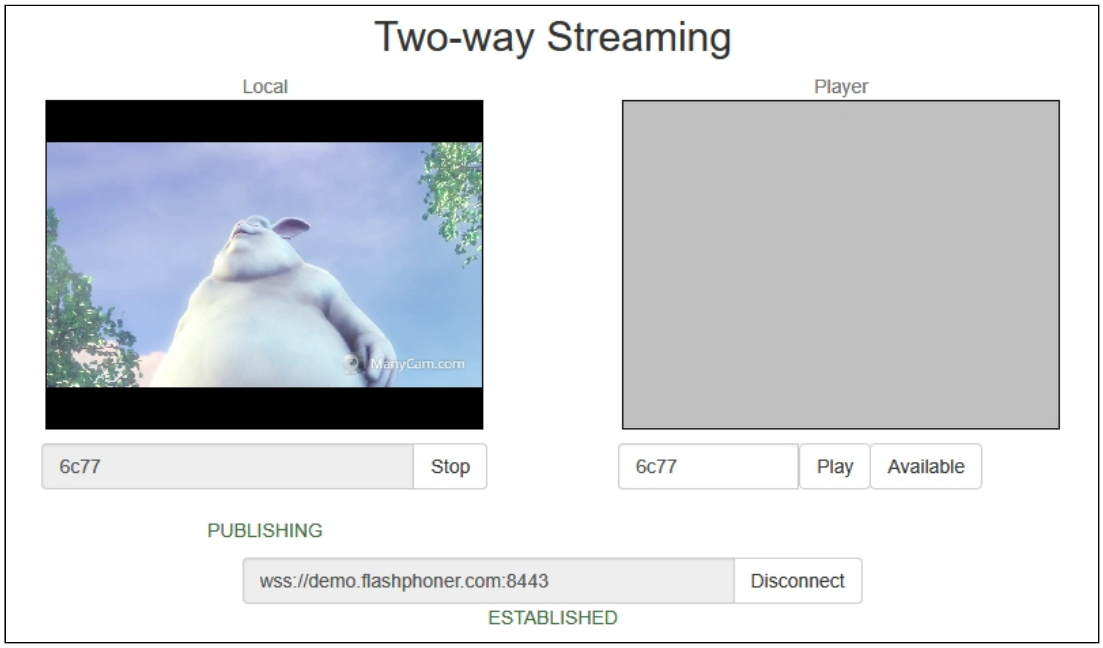
Operation flowchart



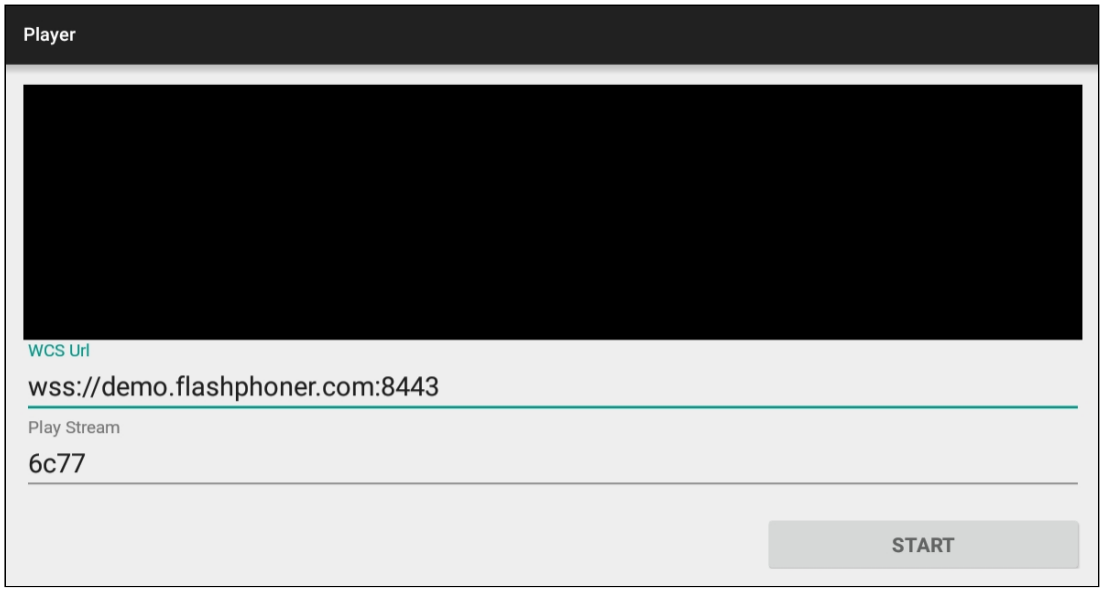
1. The browser connects to the server via the Websocket protocol and sends the `publishStream` command.
2. The browser captures the microphone and the camera and sends the WebRTC stream to the server.
3. The Android device connects to the server via the Websocket protocol and sends the `playStream` command.
4. The Android device receives the WebRTC stream from the server and plays it in the application.

Quick manual on testing

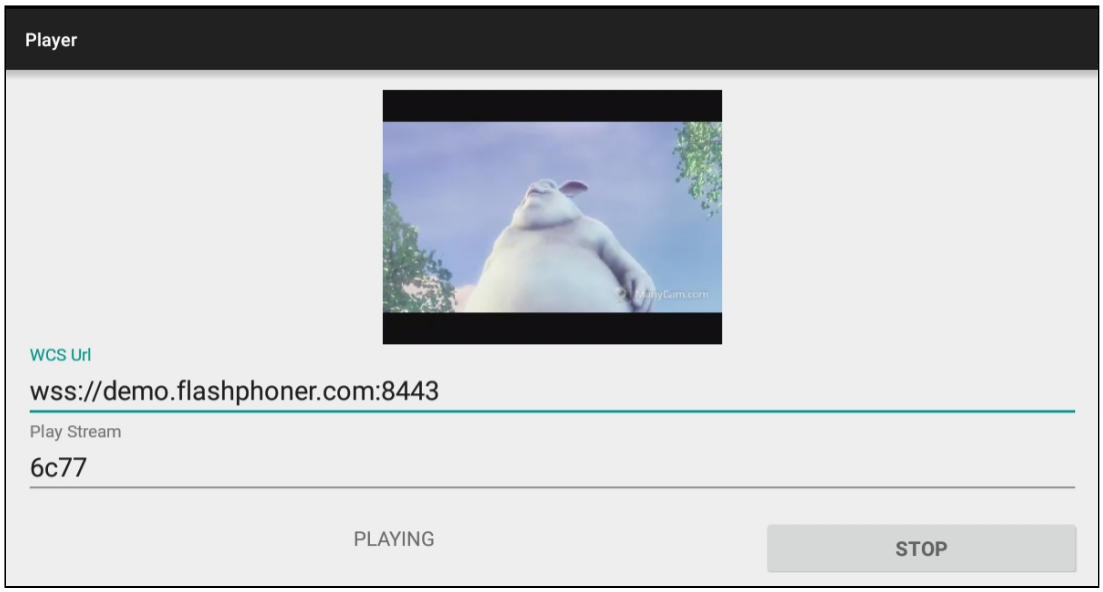
1. For the test we use:
2. the demo server at `demo.flashphoner.com`;
3. the Two Way Streaming web application to publish the stream;
4. the Player mobile application (Google Play) to play the stream
5. Open the Two Way Streaming web application. Click `Connect`, then `Publish`. Copy the identifier of the stream:



6. Install on the Android device the Player mobile app from Google Play. Start the app on the device, enter the address of the WCS server in the `WCS url` field as `wss://demo.flashphoner.com:8443`, enter the identifier of the video stream in the `Play Stream` field:



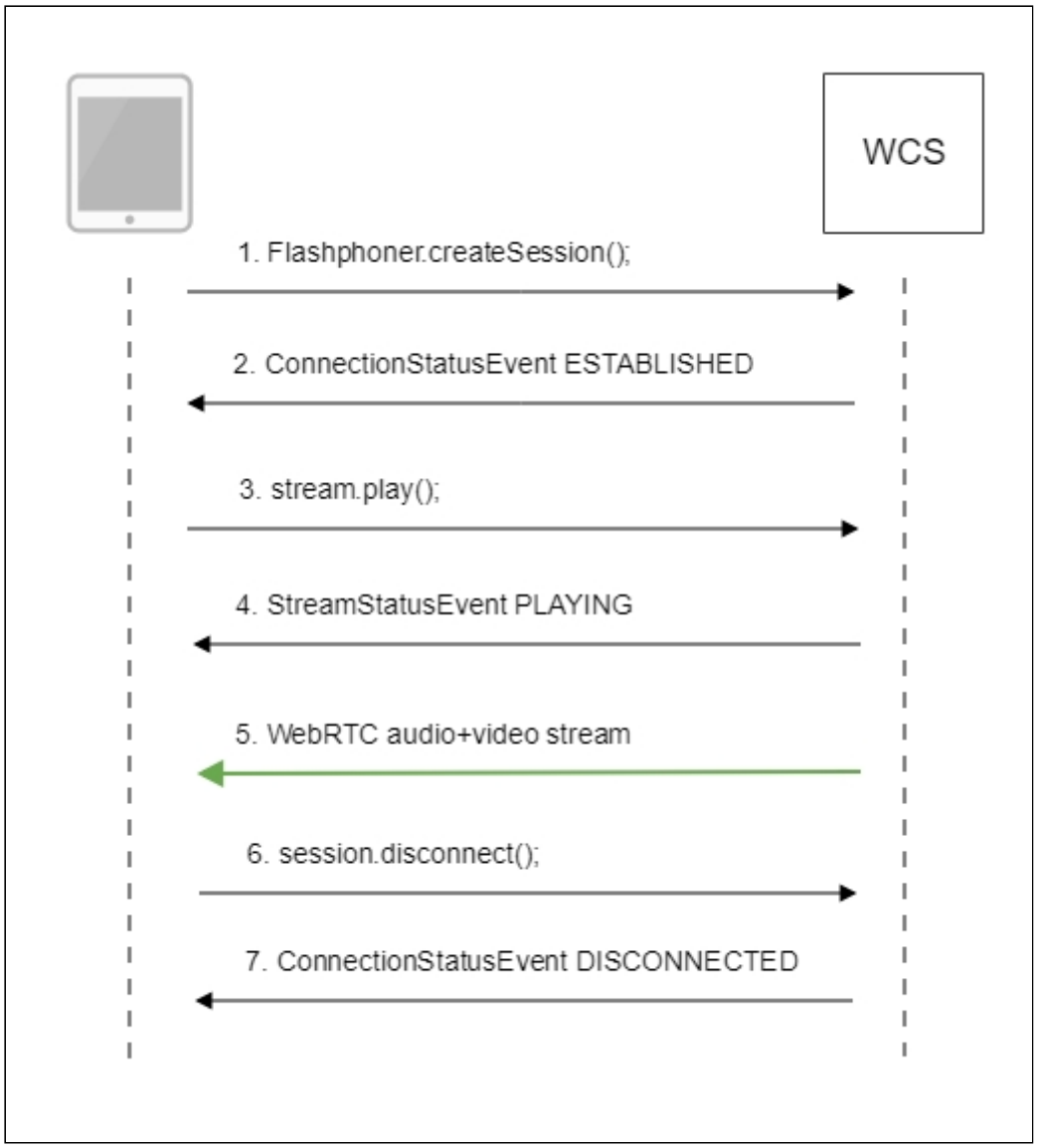
7. Click `Start`. The video stream starts playing:



Call flow

Below is the call flow when using the Player example to play the stream.

PlayerActivity.java



1. Establishing a connection to the server
   `Flashphoner.createSession()` code

```
/**
* The options for connection session are set.
* WCS server URL is passed when SessionOptions object is created.
* SurfaceViewRenderer to be used to display the video stream is set with
method SessionOptions.setRemoteRenderer().
*/
SessionOptions sessionOptions = new
SessionOptions(mWcsUrlView.getText().toString());
sessionOptions.setRemoteRenderer(remoteRender);

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

2. Receiving from the server an event that confirms 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 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);
            ...
        }
        ...
    });
    ...
}
```

3. Playing the stream
`Stream.play()` code

```
/*
 * Method Stream.play() is called to start playback of the stream.
 */
playStream.play();
```

4. Receiving from the server an event confirming successful playing of the stream
`StreamStatus.PLAYING` code

```
/**
 * Callback function for stream status change is added to display the
status.
 */
playStream.on(new StreamStatusEvent() {
    @Override
    public void onStreamStatus(final Stream stream, final StreamStatus
streamStatus) {
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
```

```
                if (!StreamStatus.PLAYING.equals(streamStatus)) {
                    Log.e(TAG, "Can not play stream " + stream.getName() +
" " + streamStatus);
                    mStatusView.setText(streamStatus.toString());
                } else if
(StreamStatus.NOT_ENOUGH_BANDWIDTH.equals(streamStatus)) {
                    Log.w(TAG, "Not enough bandwidth stream " +
stream.getName() + ", consider using lower video resolution or bitrate. "
+
                            "Bandwidth " +
(Math.round(stream.getNetworkBandwidth() / 1000)) + " " +
                            "bitrate " +
(Math.round(stream.getRemoteBitrate() / 1000)));
                } else {
                    mStatusView.setText(streamStatus.toString());
                }
            }
        });
    }
});
```

5. Receiving the audio-video stream via WebRTC
6. Stopping the playback of 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 an event confirming the playback of the stream is stopped
   `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());
        }
    });
}
```