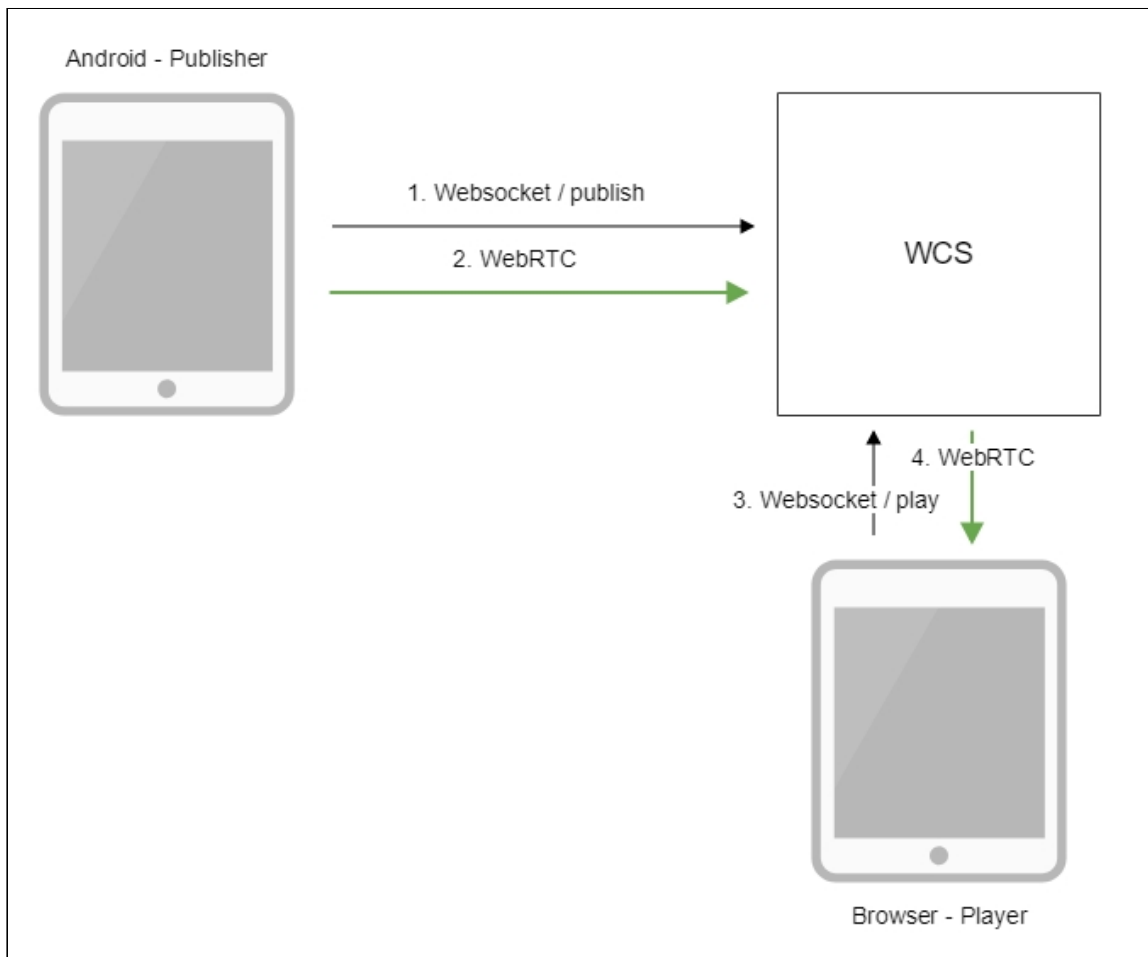


С мобильного приложения Android по WebRTC

Описание

WCS предоставляет SDK для разработки клиентских приложений на платформе Android

Схема работы

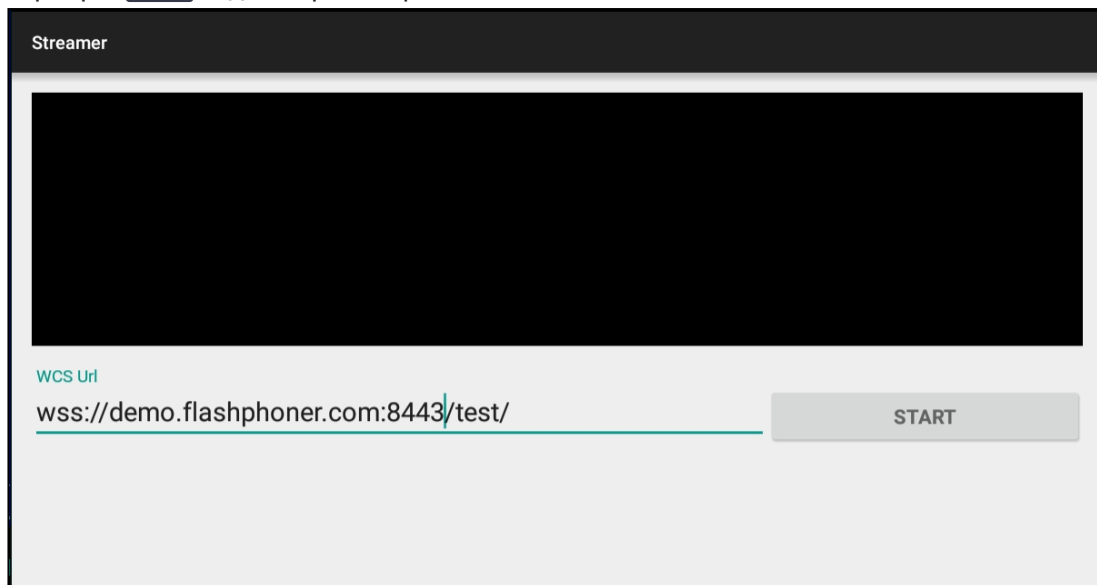


1. Android-устройство соединяется с сервером по протоколу Websocket и отправляет команду `publishStream`.
2. Android-устройство захватывает микрофон и камеру и отправляет WebRTC поток на сервер.

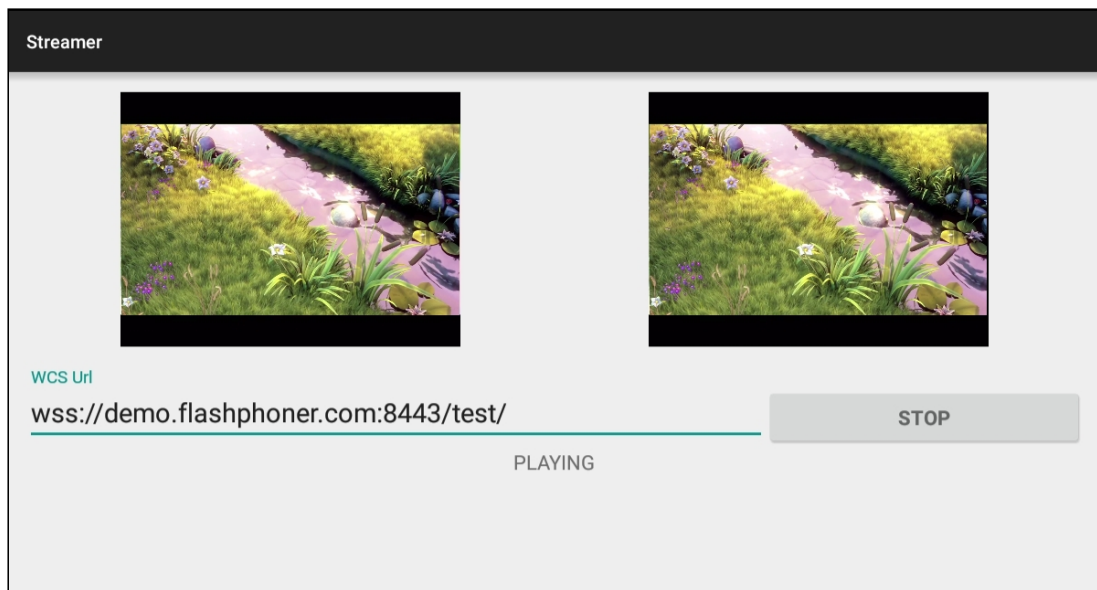
3. Браузер устанавливает соединение по Websocket и отправляет команду `playStream`.
4. Браузер получает WebRTC поток и воспроизводит этот поток на странице.

Краткое руководство по тестированию

1. Для теста используем:
2. демо-сервер `demo.flashphoner.com`;
3. мобильное приложение Streamer ([Google Play](#));
4. веб-приложение [Player](#) для отображения захваченного потока
5. Установите на Android-устройство мобильное приложение Streamer из [Google Play](#).
Запустите приложение на устройстве, введите URL в виде `wss://demo.flashphoner.com:8443/test/`, где `demo.flashphoner.com` - адрес WCS-сервера, `test` - идентификатор потока:



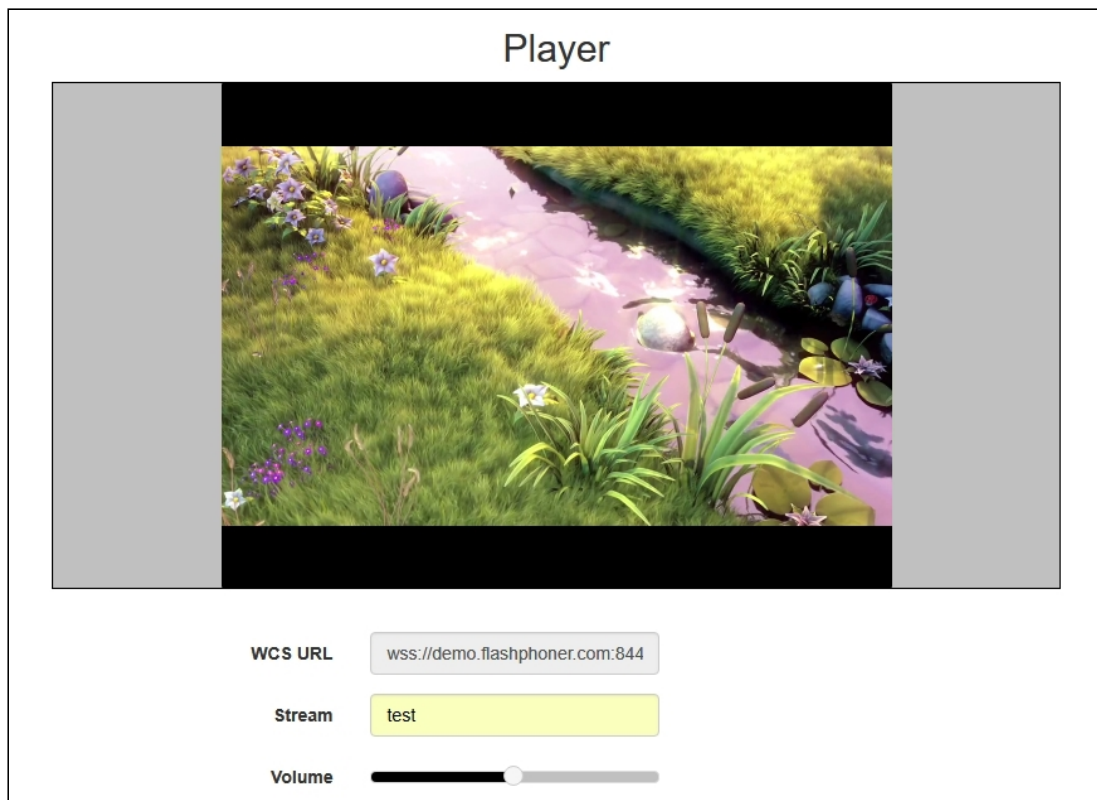
6. Нажмите кнопку `Start`. Начнется захват видеопотока с фронтальной камеры устройства и трансляция его на сервер:



7. Откройте веб-приложение Player, укажите идентификатор потока **test** в поле **Stream**:



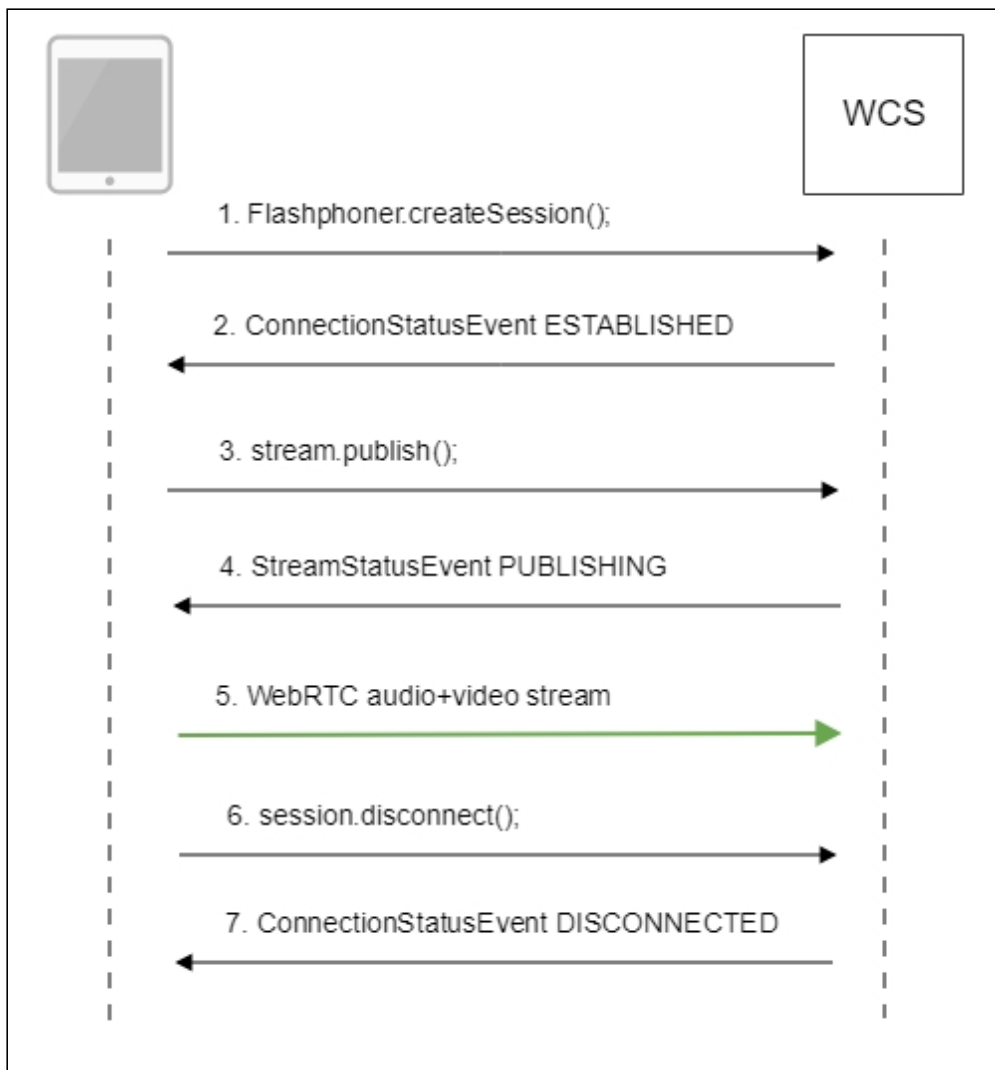
8. Нажмите кнопку **Start**. Начнется отображение трансляции с мобильного устройства:



Последовательность выполнения операций

Ниже описана последовательность вызовов при использовании примера Streamer

[StreamerActivity.java](#)



1. Установка соединения с сервером

`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 = Flashphoner.createSession(sessionOptions);

```

2. Получение от сервера события, подтверждающего успешное соединение

`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. Публикация потока

`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. Получение от сервера события, подтверждающего успешную публикацию потока

`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. Отправка аудио-видео потока по WebRTC

6. Остановка публикации потока

`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. Получение от сервера события, подтверждающего остановку публикации потока

`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());
        }
    });
}

```