

# Android Two-way Streaming

## Пример Android-приложения с плеером и стримером

Данное приложение может использоваться для публикации WebRTC-видеопотока и воспроизведения любого из следующих типов потоков с Web Call Server:

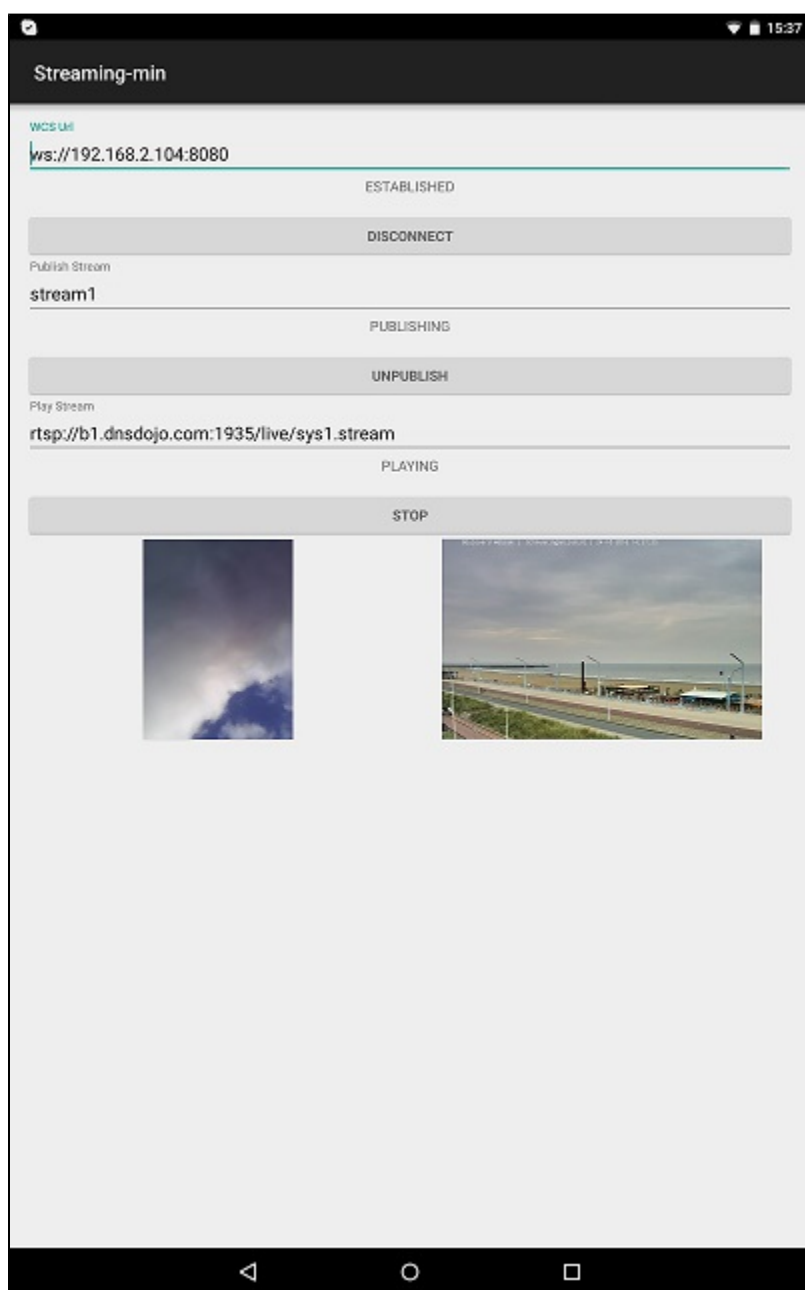
- RTSP
- WebRTC
- RTMP
- RTMFP

На скриншоте ниже представлен пример во время публикации и воспроизведения двух разных потоков.

Поля ввода

- 'WCS URL', где 192.168.2.104 - адрес WCS-сервера
- 'Publish Stream' - для имени публикуемого потока
- 'Play Stream' - для имени воспроизводимого потока

Слева отображается видео с камеры, справа воспроизводится другой поток.



## Работа с кодом примера

Для разбора кода возьмем класс [StreamingMinActivity.java](#) примера streaming-min, который доступен для скачивания в соответствующей сборке [1.0.1.38](#).

### 1. Инициализация API.

Flashphoner.init() [код](#)

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

При инициализации методу init() передается объект Context.

### 2. Создание сессии

Flashphoner.createSession() [код](#)

Методу передается объект SessionOptions со следующими параметрами

- URL WCS-сервера
- SurfaceViewRenderer localRenderer, который будет использоваться для отображения видео с камеры
- SurfaceViewRenderer remoteRenderer, который будет использоваться для отображения воспроизводимого потока

```
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. Подключение к серверу.

Session.connect() [код](#)

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

### 4. Получение от сервера события, подтверждающего успешное соединение.

session.onConnected() [код](#)

```

@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. Запрос прав на публикацию потока при нажатии Publish

ActivityCompat.requestPermissions() [код](#)

```

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. Публикация потока после предоставления соответствующих прав

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

```

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. Воспроизведение потока при нажатии Play

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

```

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. Остановка воспроизведение потока при нажатии Stop

Stream.stop() [код](#)

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

#### 9. Остановка публикации потока при нажатии Unpublish

Stream.stop() [код](#)

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

#### 10. Закрытие соединения.

Session.disconnect() [код](#)

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

#### 11. Получение события, подтверждающего разъединение.

session.onDisconnection() [код](#)

```
@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("");  
        }  
    });  
}
```