Публикация потока из приложения Android в фоновом режиме

- Описание
- Пример модификации исходного кода приложения
- Известные ограничения

Описание

Чтобы приложение Android не выгружалось из памяти устройства, и публикация видео не останавливалась при сворачивании приложения, необходимо устанавливать соединение с WCS сервером и публиковать видео из сервиса, который должен быть запущен из Activity приложения. В свою очередь, чтобы и сервис не был выгружен из памяти, необходимо создать уведомление, которое должно находиться в панели уведомлений, пока сервис работает. Рассмотрим пример модификации исходного кода приложенияAndroid Two Way Streaming

Пример модификации исходного кода приложения

1. Создание сервиса при нажатии кнопки Publish и успешном получении доступа к камере и микрофону

```
    @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) {
                    Log.i(TAG, "Permission has been denied by user");
                } else {
                    mPublishButton.setEnabled(false);
                    ...
                    Intent intent = new Intent(StreamingMinActivity.this, TestService.class);
                    intent.putExtra("url", mWcsUrlView.getText().toString());
                    intent.putExtra("streamName", mPublishStreamView.getText().toString());
                    startService(intent);

                    Log.i(TAG, "Permission has been granted by user");
                }
            }
        }
    }
```

2. Создание сессии и публикация потока при старте сервиса

```
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        SessionOptions sessionOptions = new SessionOptions(intent.getStringExtra("url"));
        Session session = Flashphoner.createSession(sessionOptions);
        session.connect(new Connection());
        StreamOptions streamOptions = new StreamOptions(intent.getStringExtra("streamName"));
        Stream publishStream = session.createStream(streamOptions);
        publishStream.publish();
        Toast.makeText(this, "Start service", Toast.LENGTH_SHORT).show();
        return START_STICKY;
    }
```

3.Создание уведомления

```
    private void showNotification() {
        Intent notificationIntent = new Intent(this, StreamingMinActivity.class);
        notificationIntent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0);
        int iconId = R.mipmap.ic_launcher;
        int uniqueCode = new Random().nextInt(Integer.MAX_VALUE);
        Notification notification = new NotificationCompat.Builder(this)
                .setSmallIcon(iconId)
                .setContentText("Started stream")
                .setContentIntent(pendingIntent).build();
        startForeground(uniqueCode, notification);
    }
```

4. Остановка сервиса при нажатии кнопки Unpublish

```
        mPublishButton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View view) {
                if (mPublishButton.getTag() == null || Integer.valueOf(R.string.action_publish).equals
(mPublishButton.getTag())) {
                    ...
                } else {
                    mPublishButton.setEnabled(false);
                    ...
                    stopService(new Intent(StreamingMinActivity.this, TestService.class));
                    publishStream = null;
                }
                ...
            }
        });
```

5. Остановка публикации при остановке сервиса

```
    @Override
    public void onDestroy() {
        super.onDestroy();
        publishStream.stop();
        Toast.makeText(this, "Stop service",
                Toast.LENGTH_SHORT).show();
        stopForeground(true);
    }
```

Полный код примера модификации файлаStreamingMinActivity.java

Code

```
package com.flashphoner.wcsexample.streaming_min;

import android.Manifest;
import android.app.IntentService;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.pm.PackageManager;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.PixelFormat;
import android.graphics.PorterDuff;
import android.os.Bundle;
import android.support.annotation.NonNull;
import android.support.v4.app.ActivityCompat;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
```

```
import android.view.inputmethod.InputMethodManager;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import com.flashphoner.fpwcsapi.Flashphoner;
import com.flashphoner.fpwcsapi.bean.Connection;
import com.flashphoner.fpwcsapi.bean.Data;
import com.flashphoner.fpwcsapi.bean.StreamStatus;
import com.flashphoner.fpwcsapi.bean.StreamStatusInfo;
import com.flashphoner.fpwcsapi.layout.PercentFrameLayout;
import com.flashphoner.fpwcsapi.session.Session;
import com.flashphoner.fpwcsapi.session.SessionEvent;
import com.flashphoner.fpwcsapi.session.SessionOptions;
import com.flashphoner.fpwcsapi.session.Stream;
import com.flashphoner.fpwcsapi.session.StreamOptions;
import com.flashphoner.fpwcsapi.session.StreamStatusEvent;

import junit.framework.Test;

import org.webrtc.PeerConnection;
import org.webrtc.RendererCommon;
import org.webrtc.SurfaceViewRenderer;

import java.util.ArrayList;
import java.util.List;

/**
 * Example with streamer and player.
 * Demonstrates how to publish a video stream while playing another one.
 */
public class StreamingMinActivity extends AppCompatActivity {

    private static String TAG = StreamingMinActivity.class.getName();

    private static final int PUBLISH_REQUEST_CODE = 100;


    // UI references.
    private EditText mWcsUrlView;
    private TextView mConnectStatus;
    private Button mConnectButton;
    private EditText mPublishStreamView;
    private TextView mPublishStatus;
    private Button mPublishButton;
    private EditText mPlayStreamView;
    private TextView mPlayStatus;
    private Button mPlayButton;

    private Session session;

    private Stream publishStream;
    private Stream playStream;

    private SurfaceViewRenderer localRender;
    private SurfaceViewRenderer remoteRender;

    private PercentFrameLayout localRenderLayout;
    private PercentFrameLayout remoteRenderLayout;
    private SessionOptions sessionOptions;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_streaming_min);

        /**
         * Initialization of the API.
         */
```

```
        Flashphoner.init(this);

        mWcsUrlView = (EditText) findViewById(R.id.wcs_url);
        SharedPreferences sharedPref = this.getPreferences(Context.MODE_PRIVATE);
        mWcsUrlView.setText(sharedPref.getString("wcs_url", getString(R.string.wcs_url)));
        mConnectStatus = (TextView) findViewById(R.id.connect_status);
        mConnectButton = (Button) findViewById(R.id.connect_button);

        /**
         * Connection to server will be established when Connect button is clicked.
         */
        mConnectButton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View view) {
                if (mConnectButton.getTag() == null || Integer.valueOf(R.string.action_connect).equals
(mConnectButton.getTag())) {
                    /**
                     * 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 video of the played stream is set with method
SessionOptions.setRemoteRenderer().
                     */
                    sessionOptions = new SessionOptions(mWcsUrlView.getText().toString());
                    sessionOptions.setLocalRenderer(localRender);
                    sessionOptions.setRemoteRenderer(remoteRender);

                    /**
                     * Uncomment this code to use your own RTCConfiguration. 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.RTCConfiguration customConfig = new PeerConnection.RTCConfiguration
(iceServers);
                    //sessionOptions.setMediaOptions(customConfig);

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

                    /**
                     * Callback functions for session status events are added to make appropriate changes in
controls of the interface when connection is established and closed.
                     */
                    session.on(new SessionEvent() {
                        @Override
                        public void onAppData(Data data) {

                        }

                        @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);
                                }
                            });
                        }

                        @Override
                        public void onRegistered(Connection connection) {
```

```
                        }

                        @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("");
                                }
                            });
                        }
                    });
                    mConnectButton.setEnabled(false);

                    /**
                     * Connection to WCS server is established with method Session.connect().
                     */
                    session.connect(new Connection());

                    SharedPreferences sharedPref = StreamingMinActivity.this.getPreferences(Context.
MODE_PRIVATE);
                    SharedPreferences.Editor editor = sharedPref.edit();
                    editor.putString("wcs_url", mWcsUrlView.getText().toString());
                    editor.apply();
                } else {
                    mConnectButton.setEnabled(false);

                    /**
                     * Connection to WCS server is closed with method Session.disconnect().
                     */
                    session.disconnect();
                }
                View currentFocus = getCurrentFocus();
                if (currentFocus != null) {
                    InputMethodManager inputManager = (InputMethodManager) getSystemService(Context.
INPUT_METHOD_SERVICE);
                    inputManager.hideSoftInputFromWindow(currentFocus.getWindowToken(), InputMethodManager.
HIDE_NOT_ALWAYS);
                }
            }
        });

        mPublishStreamView = (EditText) findViewById(R.id.publish_stream);
        mPublishStreamView.setText(sharedPref.getString("publish_stream", getString(R.string.
default_publish_name)));
        mPublishStatus = (TextView) findViewById(R.id.publish_status);
        mPublishButton = (Button) findViewById(R.id.publish_button);

        /**
         * Stream will be published when Publish button is clicked.
         */
        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);
```

```
                    SharedPreferences sharedPref = StreamingMinActivity.this.getPreferences(Context.
MODE_PRIVATE);
                    SharedPreferences.Editor editor = sharedPref.edit();
                    editor.putString("publish_stream", mPublishStreamView.getText().toString());
                    editor.apply();
                } else {
                    mPublishButton.setEnabled(false);
                    /**
                     * Method Stream.stop() is called to unpublish the stream.
                     */
                    //publishStream.stop();
                    stopService(new Intent(StreamingMinActivity.this, TestService.class));
                    publishStream = null;
                }
                View currentFocus = getCurrentFocus();
                if (currentFocus != null) {
                    InputMethodManager inputManager = (InputMethodManager) getSystemService(Context.
INPUT_METHOD_SERVICE);
                    inputManager.hideSoftInputFromWindow(currentFocus.getWindowToken(), InputMethodManager.
HIDE_NOT_ALWAYS);
                }
            }
        });

        mPlayStreamView = (EditText) findViewById(R.id.play_stream);
        mPlayStreamView.setText(sharedPref.getString("play_stream", getString(R.string.default_play_name)));
        mPlayStatus = (TextView) findViewById(R.id.play_status);
        mPlayButton = (Button) findViewById(R.id.play_button);

        /**
         * Stream playback will be started when Play button is clicked.
         */
        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);

                    /**
                     * Callback function for stream status change is added to make appropriate changes in
controls of the interface when playing.
                     */
                    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)) {
                                        mPlayButton.setText(R.string.action_stop);
                                        mPlayButton.setTag(R.string.action_stop);
                                    } 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 {
```

```
                                        mPlayButton.setText(R.string.action_play);
                                        mPlayButton.setTag(R.string.action_play);
                                    }
                                    mPlayButton.setEnabled(true);
                                    if (StreamStatus.FAILED.equals(streamStatus)){
                                        switch (stream.getInfo()){
                                            case StreamStatusInfo.SESSION_DOES_NOT_EXIST:
                                                mPlayStatus.setText(streamStatus+": Actual session does not
exist");
                                                break;
                                            case StreamStatusInfo.STOPPED_BY_PUBLISHER_STOP:
                                                mPlayStatus.setText(streamStatus+": Related publisher stopped
its stream or lost connection");
                                                break;
                                            case StreamStatusInfo.SESSION_NOT_READY:
                                                mPlayStatus.setText(streamStatus+": Session is not initialized
or terminated on play ordinary stream");
                                                break;
                                            case StreamStatusInfo.RTSP_STREAM_NOT_FOUND:
                                                mPlayStatus.setText(streamStatus+": Rtsp stream not found where
agent received '404-Not Found'");
                                                break;
                                            case StreamStatusInfo.FAILED_TO_CONNECT_TO_RTSP_STREAM:
                                                mPlayStatus.setText(streamStatus+": Failed to connect to rtsp
stream");
                                                break;
                                            case StreamStatusInfo.FILE_NOT_FOUND:
                                                mPlayStatus.setText(streamStatus+": File does not exist, check
filename");
                                                break;
                                            case StreamStatusInfo.FILE_HAS_WRONG_FORMAT:
                                                mPlayStatus.setText(streamStatus+": File has wrong format on
play vod, this format is not supported");
                                                break;
                                            default:{
                                               mPlayStatus.setText(stream.getInfo());
                                              }
                                        }
                                    } else {
                                        mPlayStatus.setText(streamStatus.toString());
                                    }
                                }
                            });
                        }
                    });

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

                    SharedPreferences sharedPref = StreamingMinActivity.this.getPreferences(Context.
MODE_PRIVATE);
                    SharedPreferences.Editor editor = sharedPref.edit();
                    editor.putString("play_stream", mPlayStreamView.getText().toString());
                    editor.apply();
                } else {
                    /**
                     * Method Stream.stop() is called to stop playback of the stream.
                     */
                    playStream.stop();
                    playStream = null;
                }
                View currentFocus = getCurrentFocus();
                if (currentFocus != null) {
                    InputMethodManager inputManager = (InputMethodManager) getSystemService(Context.
INPUT_METHOD_SERVICE);
                    inputManager.hideSoftInputFromWindow(currentFocus.getWindowToken(), InputMethodManager.
HIDE_NOT_ALWAYS);
                }
            }
```

```
        });

        localRender = (SurfaceViewRenderer) findViewById(R.id.local_video_view);
        remoteRender = (SurfaceViewRenderer) findViewById(R.id.remote_video_view);

        localRenderLayout = (PercentFrameLayout) findViewById(R.id.local_video_layout);
        remoteRenderLayout = (PercentFrameLayout) findViewById(R.id.remote_video_layout);

        localRender.setZOrderMediaOverlay(true);

        remoteRenderLayout.setPosition(0, 0, 100, 100);
        remoteRender.setScalingType(RendererCommon.ScalingType.SCALE_ASPECT_FIT);
        remoteRender.setMirror(false);
        remoteRender.requestLayout();

        localRenderLayout.setPosition(0, 0, 100, 100);
        localRender.setScalingType(RendererCommon.ScalingType.SCALE_ASPECT_FIT);
        localRender.setMirror(true);
        localRender.requestLayout();
    }

    @Override
    protected void onResume() {
        super.onResume();
        try {
            localRender.init(Flashphoner.context, null);
        } catch (IllegalStateException e) {
            //ignore
        }
        try {
            remoteRender.init(Flashphoner.context, null);
        } catch (IllegalStateException e) {
            //ignore
        }

    }

    @Override
    protected void onPause() {
        super.onPause();
        localRender.release();
        remoteRender.release();
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        if (session != null) {
            session.disconnect();
        }
    }

    @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) {
                    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);

                    /**
                     * Callback function for stream status change is added to make appropriate changes in
controls of the interface when publishing.
                     */
                    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)) {
                                        mPublishButton.setText(R.string.action_unpublish);
                                        mPublishButton.setTag(R.string.action_unpublish);
                                    } else {
                                        mPublishButton.setText(R.string.action_publish);
                                        mPublishButton.setTag(R.string.action_publish);
                                    }
                                    mPublishButton.setEnabled(true);
                                    if (StreamStatus.FAILED.equals(streamStatus)){
                                        switch (stream.getInfo()){
                                            case StreamStatusInfo.STREAM_NAME_ALREADY_IN_USE:
                                                mPublishStatus.setText(streamStatus+": Server already has a
publish stream with the same name, try using different one");
                                                break;
                                            default:{
                                                mPlayStatus.setText(stream.getInfo());
                                            }
                                        }
                                    } else {
                                        mPublishStatus.setText(streamStatus.toString());
                                    }
                                }
                            });
                        }
                    });

                    /**
                     * Method Stream.publish() is called to publish stream.
                     */
                    //publishStream.publish();
                    Intent intent = new Intent(StreamingMinActivity.this, TestService.class);
                    intent.putExtra("url", mWcsUrlView.getText().toString());
                    intent.putExtra("streamName", mPublishStreamView.getText().toString());
                    startService(intent);

                    Log.i(TAG, "Permission has been granted by user");
                }
            }
        }
    }
}
```

Полный код примера файла реализации сервисаTestService.java

Code

```
package com.flashphoner.wcsexample.streaming_min;

import android.app.Activity;
import android.app.Notification;
```

```
import android.app.PendingIntent;
import android.app.Service;
import android.content.Intent;
import android.media.MediaPlayer;
import android.os.IBinder;
import android.support.v4.app.NotificationCompat;
import android.widget.EditText;
import android.widget.Toast;

import com.flashphoner.fpwcsapi.Flashphoner;
import com.flashphoner.fpwcsapi.bean.Connection;
import com.flashphoner.fpwcsapi.session.Session;
import com.flashphoner.fpwcsapi.session.SessionOptions;
import com.flashphoner.fpwcsapi.session.Stream;
import com.flashphoner.fpwcsapi.session.StreamOptions;

import junit.framework.Test;

import java.util.ArrayList;
import java.util.Random;

public class TestService extends Service {

    private static Stream publishStream;
    private static Session session;

    @Override
    public IBinder onBind(Intent intent) {
        // TODO: Return the communication channel to the service.
        //throw new UnsupportedOperationException("Not yet implemented");
        return null;
    }

    @Override
    public void onCreate() {
        super.onCreate();
        showNotification();
        Toast.makeText(this, "Create service",
                Toast.LENGTH_SHORT).show();
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        SessionOptions sessionOptions = new SessionOptions(intent.getStringExtra("url"));
        Session session = Flashphoner.createSession(sessionOptions);
        session.connect(new Connection());
        StreamOptions streamOptions = new StreamOptions(intent.getStringExtra("streamName"));
        Stream publishStream = session.createStream(streamOptions);
        publishStream.publish();
        Toast.makeText(this, "Start service", Toast.LENGTH_SHORT).show();
        return START_STICKY;
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        publishStream.stop();
        Toast.makeText(this, "Stop service",
                Toast.LENGTH_SHORT).show();
        stopForeground(true);
    }

    private void showNotification() {
        Intent notificationIntent = new Intent(this, StreamingMinActivity.class);
        notificationIntent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0);
        int iconId = R.mipmap.ic_launcher;
        int uniqueCode = new Random().nextInt(Integer.MAX_VALUE);
        Notification notification = new NotificationCompat.Builder(this)
                .setSmallIcon(iconId)
                .setContentText("Started stream")
```

```
                .setContentIntent(pendingIntent).build();
        startForeground(uniqueCode, notification);
    }

    public static void setSession(Session session) {
        TestService.session = session;
    }

    public static void setPublishStream(Stream publishStream) {
        TestService.publishStream = publishStream;
    }
}
```

Полный код примера модификации манифеста приложения

Code

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.flashphoner.wcsexample.streaming_min">

    <uses-feature android:name="android.hardware.camera" />
    <uses-feature android:name="android.hardware.camera.autofocus" />
    <uses-feature
        android:glEsVersion="0x00020000"
        android:required="true" />

    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
    <uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".StreamingMinActivity"
            android:configChanges="orientation|screenSize"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service
            android:name=".TestService"
            android:enabled="true"
            android:exported="true">
        </service>
    </application>

</manifest>
```

Известные ограничения

1. При публикации потока из сервиса невозможно отображение локального видео в приложении

2. В приведенном примере, при закрытии приложения из списка запущенных приложений, сервис также остановится.