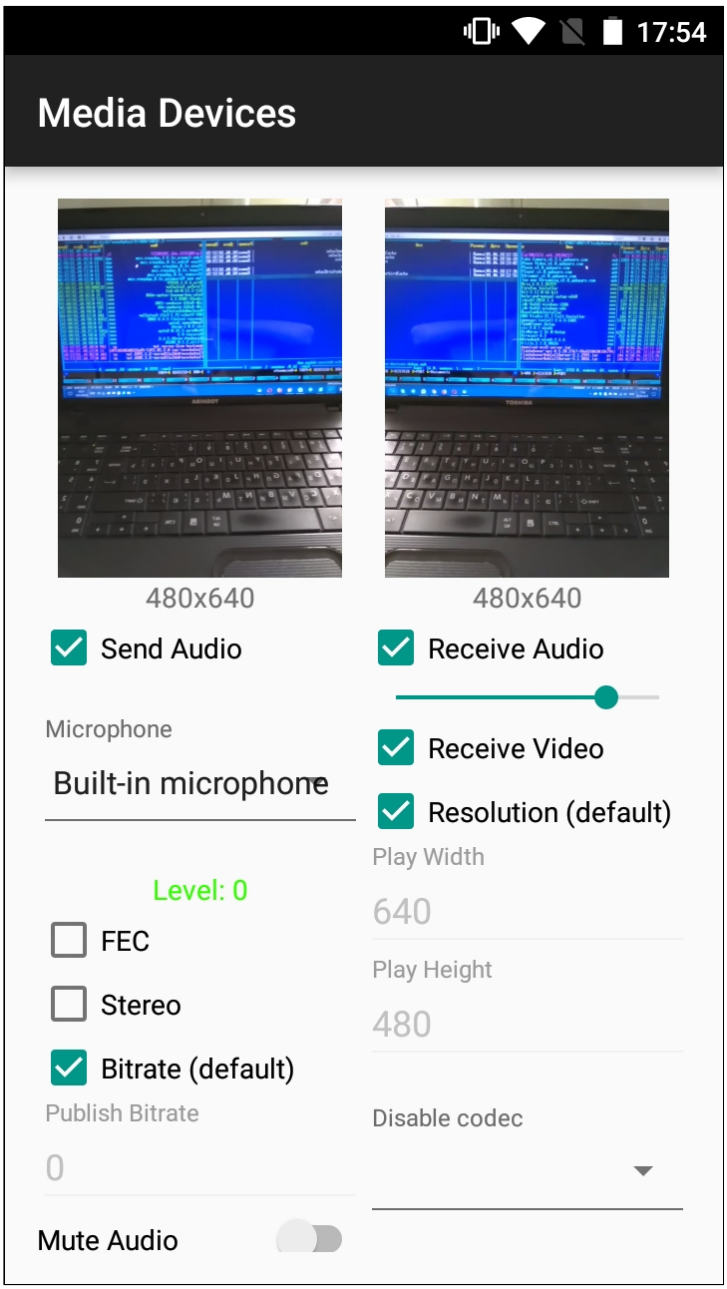
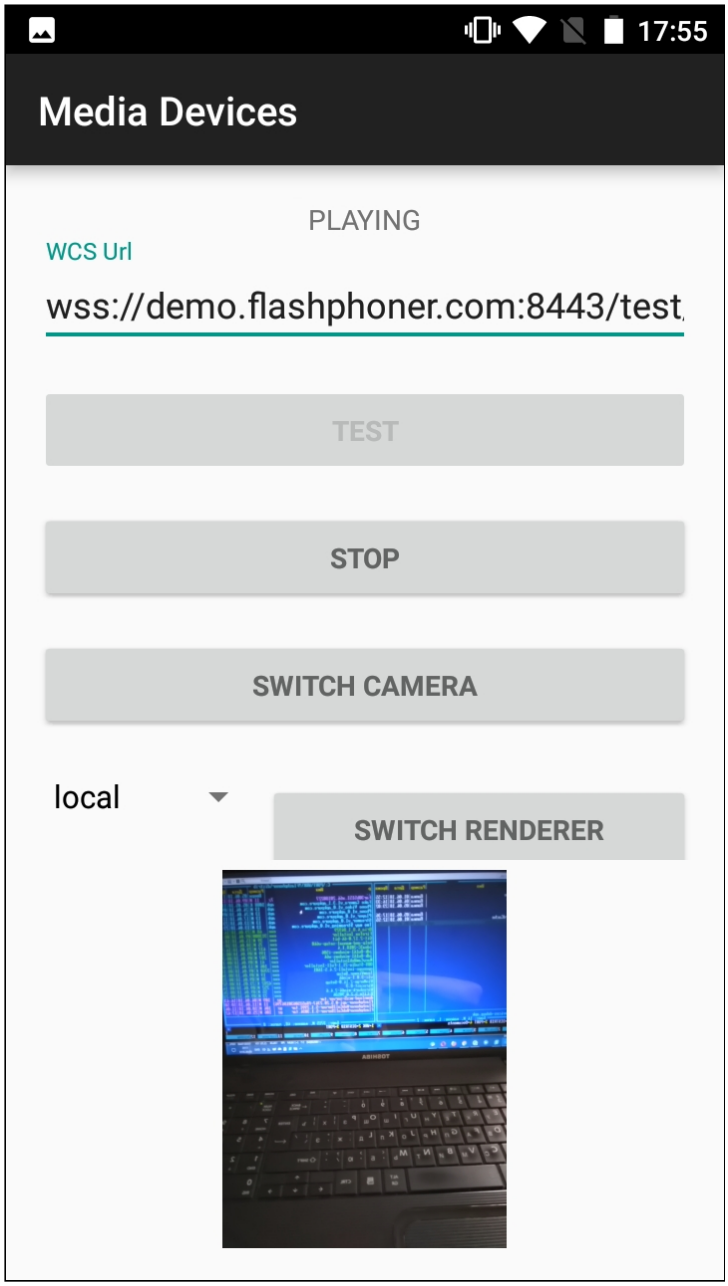
Android Media Devices

Пример Android-приложения для управления медиа-устройствами

Данный пример может использоваться как стример для публикации WebRTC-видеопотока на Web Call Server и позволяет выбрать медиа-устройства и параметры для публикуемого видео



Пример переключения объекта для вывода изображения с камеры



Работа с кодом примера

Для разбора кода возьмем класс MediaDevicesActivity.java примера `media-devices`, который доступен для скачивания в соответствующей сборке 1.0.1.70.

1. Инициализация API

`Flashphoner.init()` code

При инициализации методу `init()` передается объект `Context`.

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

2. Получение списка доступных медиа-устройств

`Flashphoner.getMediaDevices()`, `MediaDeviceList.getAudioList()`, `MediaDeviceList.getVideoList()` code

```
mMicSpinner = (LabelledSpinner) findViewById(R.id.microphone);
mMicSpinner.setItemsArray(Flashphoner.getMediaDevices().getAudioList());

mMicLevel = (TextView) findViewById(R.id.microphone_level);

mCameraSpinner = (LabelledSpinner) findViewById(R.id.camera);
mCameraSpinner.setItemsArray(Flashphoner.getMediaDevices().getVideoList());
```

3. Управление отображением видео

`FPSurfaceViewRenderer.setMirror()` code

При показе видео используются следующие объекты `FPSurfaceViewRenderer`:

- `localRender` для отображения видео с камеры
- `remoteRender` для отображения публикуемого потока
- `newSurfaceRenderer` для демонстрации переключения объекта

Для этих объектов устанавливается позиция на экране, тип масштабирования и зеркалирование.

По умолчанию, для отображения видео с камеры устанавливается зеркальная ориентация при помощи метода `setMirror(true)`. Для отображения публикуемого потока и объекта для демонстрации переключения зеркалирование отключается при помощи `setMirror(false)`:

```
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();

switchRenderLayout.setPosition(0, 0, 100, 100);
newSurfaceRenderer.setZOrderMediaOverlay(true);
newSurfaceRenderer.setScalingType(RendererCommon.ScalingType.SCALE_ASPECT_FIT);

newSurfaceRenderer.setMirror(true);
newSurfaceRenderer.requestLayout();
```

В данном случае, при выборе фронтальной камеры изображение с камеры выглядит нормально, но публикуется зеркальным. При выборе тыловой камеры изображение с камеры будет выглядеть зеркальным, а публикуемый поток будет иметь нормальную ориентацию (см. скриншоты приложения выше).

4. Настройка параметров аудио и видео, заданных пользователем

`AudioConstraints`, `VideoConstraints` code

```
@NonNull
private Constraints getConstraints() {
    AudioConstraints audioConstraints = null;
    if (mSendAudio.isChecked()) {
        audioConstraints = new AudioConstraints();
        if (mUseFEC.isChecked()) {
            audioConstraints.setUseFEC(true);
        }
        if (mUseStereo.isChecked()) {
            audioConstraints.setUseStereo(true);
        }
        if (!mDefaultPublishAudioBitrate.isChecked() &&
mDefaultPublishAudioBitrate.getText().length() > 0) {

audioConstraints.setBitrate(Integer.parseInt(mPublishAudioBitrate.getText().toSt

        }
    }
    VideoConstraints videoConstraints = null;
    if (mSendVideo.isChecked()) {
        videoConstraints = new VideoConstraints();
        videoConstraints.setCameraId(((MediaDevice)
mCameraSpinner.getSpinner().getSelectedItem()).getId());
        if (mCameraFPS.getText().length() > 0) {

videoConstraints.setVideoFps(Integer.parseInt(mCameraFPS.getText().toString()));

        }
        if (mWidth.getText().length() > 0 && mHeight.getText().length() > 0)
{

videoConstraints.setResolution(Integer.parseInt(mWidth.getText().toString()),
                    Integer.parseInt(mHeight.getText().toString()));
        }
        if (!mDefaultPublishVideoBitrate.isChecked() &&
mPublishVideoBitrate.getText().length() > 0) {

videoConstraints.setBitrate(Integer.parseInt(mPublishVideoBitrate.getText().toSt

        }
    }
    return new Constraints(audioConstraints, videoConstraints);
}
```

5. Локальное тестирование камеры и микрофона

`Flashphoner.getLocalMediaAccess()` code

Методу передаются:

- настройки аудио и видео, заданные пользователем
- локальный объект `SurfaceViewRenderer localRenderer` для вывода изображения с выбранной камеры

```
case TEST_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 {
        Flashphoner.getLocalMediaAccess(getConstraints(), localRender);
        mTestButton.setText(R.string.action_release);
        mTestButton.setTag(R.string.action_release);
        mStartButton.setEnabled(false);
        soundMeter = new SoundMeter();
        soundMeter.start();
        ...
        Log.i(TAG, "Permission has been granted by user");
    }
break;
```

6. Создание сессии

`Flashphoner.createSession()` code

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 воспроизведения опубликованного видеопотока

```
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);
```

7. Подключение к серверу

`Session.connect()` code

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

8. Получение от сервера события, подтверждающего успешное соединение

`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);
           mTestButton.setEnabled(false);
           mStatusView.setText(connection.getStatus());;
           ...
       }
   });
}
```

9. Создание потока и подготовка к публикации

`Session.createStream()` code

```
publishStream = session.createStream(streamOptions);
if (mMuteAudio.isChecked()) {
    publishStream.muteAudio();
}
if (mMuteVideo.isChecked()) {
    publishStream.muteVideo();
}
...

ActivityCompat.requestPermissions(MediaDevicesActivity.this,
         new String[]{Manifest.permission.RECORD_AUDIO,
Manifest.permission.CAMERA},
         PUBLISH_REQUEST_CODE);
```

10. Публикация потока

`Stream.publish()` code

```
case PUBLISH_REQUEST_CODE: {
    if (grantResults.length == 0 ||
           grantResults[0] != PackageManager.PERMISSION_GRANTED ||
           grantResults[1] != PackageManager.PERMISSION_GRANTED) {
        mStartButton.setEnabled(false);
        mTestButton.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");
    }
    break;
}
```

11. Получение от сервера события, подтверждающего успешную публикацию потока

`StreamStatusEvent.PUBLISHING` code

При получении данного события создается превью-видеопоток при помощи `Session.createStream()` и вызывается `Stream.play()` для его воспроизведения.

```
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);

                    streamOptions.setConstraints(new
Constraints(mReceiveAudio.isChecked(), mReceiveVideo.isChecked()));

                    VideoConstraints videoConstraints = null;
                    if (mReceiveVideo.isChecked()) {
                        videoConstraints = new VideoConstraints();
                        ...
                    }
                    AudioConstraints audioConstraints = null;
                    if (mReceiveAudio.isChecked()) {
                        audioConstraints = new AudioConstraints();
                    }
```

```
                    streamOptions.setConstraints(new
Constraints(audioConstraints, videoConstraints));
                    String[] stripCodec = {(String)
mStripPlayerCodec.getSpinner().getSelectedItem()};
                    streamOptions.setStripCodecs(stripCodec);
                    /**
                      * Stream is created with method Session.createStream().
                      */
                    playStream = session.createStream(streamOptions);
                    ...
                    /**
                      * Method Stream.play() is called to start playback of
the stream.
                      */
                    playStream.play();
                    if (mSendVideo.isChecked())
                        mSwitchCameraButton.setEnabled(true);
                    mSwitchRendererButton.setEnabled(true);
                } else {
                    Log.e(TAG, "Can not publish stream " + stream.getName() +
" " + streamStatus);
                }
                mStatusView.setText(streamStatus.toString());
            }
        });
    }
});
```

12. Переключение камеры во время трансляции

`Stream.switchCamera()` code

```
mSwitchCameraButton.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View view) {
        if (publishStream != null) {
            mSwitchCameraButton.setEnabled(false);
            publishStream.switchCamera(new CameraSwitchHandler() {
                @Override
                public void onCameraSwitchDone(boolean var1) {
                    runOnUiThread(new Runnable() {
                        @Override
                        public void run() {
                            mSwitchCameraButton.setEnabled(true);
                        }
                    });

                }

                @Override
                public void onCameraSwitchError(String var1) {
                    runOnUiThread(new Runnable() {
                        @Override
                        public void run() {
                            mSwitchCameraButton.setEnabled(true);
```

```
                        }
                    });
                }
            });
        }
    }

});
```

13. Переключение объекта для отображения видеопотока во время трансляции

`Stream.switchRenderer()` code

```
mSwitchRendererButton.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        if (spinner.getSelectedItemId() == 0){
            if (isSwitchRemoteRenderer) {
                playStream.switchRenderer(remoteRender);
                isSwitchRemoteRenderer = false;
            }
            if (!isSwitchLocalRenderer) {
                publishStream.switchRenderer(newSurfaceRenderer);
                isSwitchLocalRenderer = true;
            } else {
                publishStream.switchRenderer(localRender);
                isSwitchLocalRenderer = false;
            }
        } else {
            if (isSwitchLocalRenderer) {
                publishStream.switchRenderer(localRender);
                isSwitchLocalRenderer = false;
            }
            if (!isSwitchRemoteRenderer) {
                playStream.switchRenderer(newSurfaceRenderer);
                isSwitchRemoteRenderer = true;
            } else {
                playStream.switchRenderer(remoteRender);
                isSwitchRemoteRenderer = false;
            }
        }
    }
});
```

14. Управление звуком при помощи аппаратных кнопок

`Flashphoner.setVolume()` code

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    int currentVolume = Flashphoner.getVolume();
    switch (keyCode) {
```

```
        case KeyEvent.KEYCODE_VOLUME_DOWN:
            if (currentVolume == 1) {
                Flashphoner.setVolume(0);
            }
            mPlayVolume.setProgress(currentVolume-1);
            break;
        case KeyEvent.KEYCODE_VOLUME_UP:
            if (currentVolume == 0) {
                Flashphoner.setVolume(1);
            }
            mPlayVolume.setProgress(currentVolume+1);
            break;
    }
    return super.onKeyDown(keyCode, event);
}
```

15. Использование внешнего динамика телефона

`Flashphoner.getAudioManager().isSpeakerphoneOn()`, `Flashphoner.getAudioManager().setUseSpeakerPhone()` code

```
mSpeakerPhone = (CheckBox) findViewById(R.id.use_speakerphone);
mSpeakerPhone.setChecked(Flashphoner.getAudioManager().getAudioManager().isSpeak

mSpeakerPhone.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean
isChecked) {
        Flashphoner.getAudioManager().setUseSpeakerPhone(isChecked);
    }
});
```

16. Закрытие соединения

`Session.disconnect()` code

```
mStartButton.setEnabled(false);
mTestButton.setEnabled(false);

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

17. Получение события, подтверждающего разъединение

`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);
           mSwitchCameraButton.setEnabled(false);
           mSwitchRendererButton.setEnabled(false);
           mStatusView.setText(connection.getStatus());
           mTestButton.setEnabled(true);
       }
   });
}
```