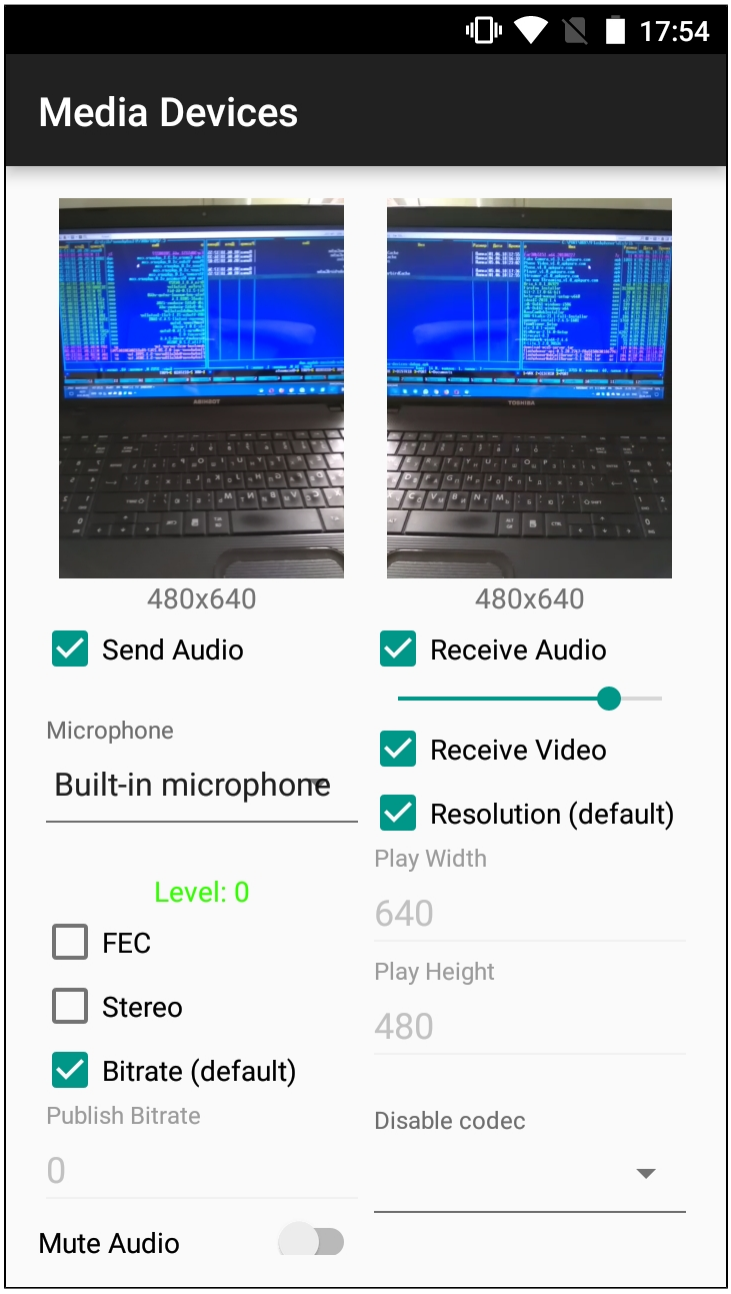
Android Media Devices

Example of Android application managing media devices

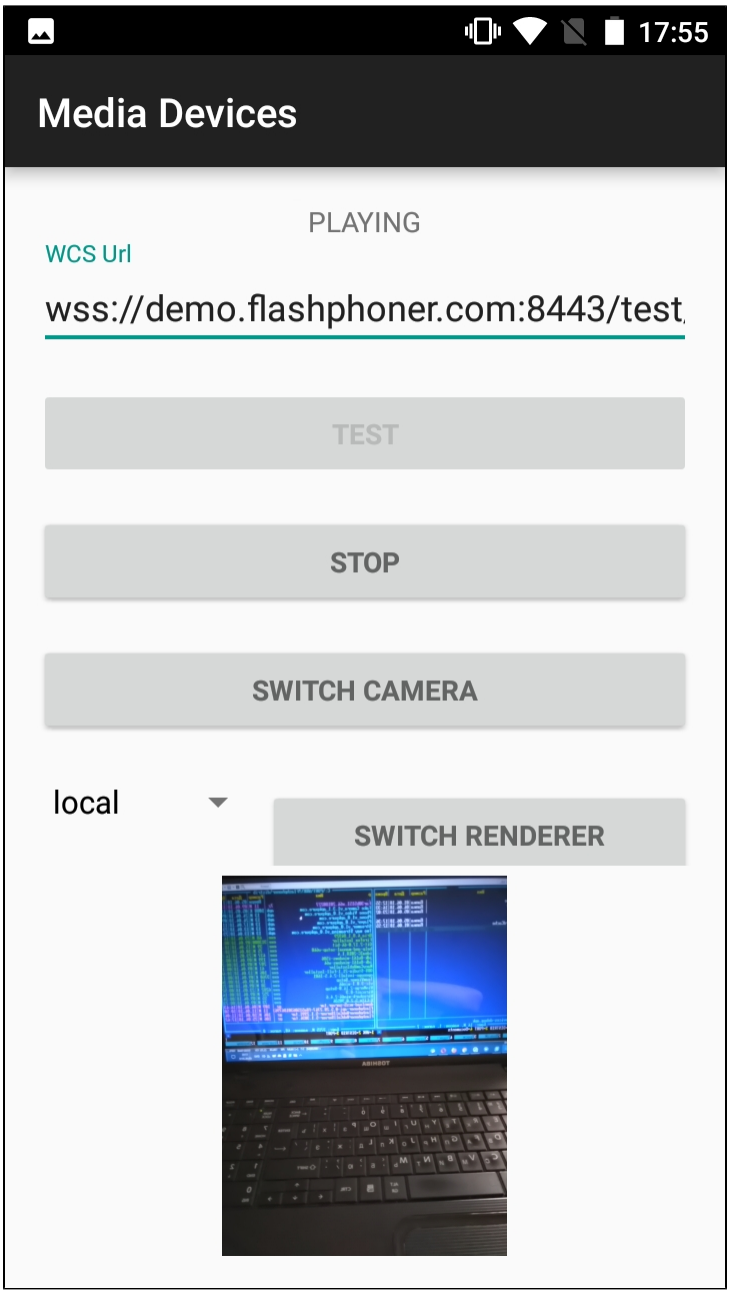
This example can be used as streamer allowing to select source camera and microphone and specify parameters for the published video: FPS (Frames Per Second) and resolution (width, height).

Below that input field are located drop-down lists of available microphones and cameras and input fields for video parameters.
Two videos are played

- left - video from the camera
- right - the published video stream as received from the server



Switching renderer to play video stream from camera:



Work with code of the example

To analyze the code, let's take class MediaDevicesActivity.javaof the media-devices example, which can be downloaded with corresponding build 1.0.1.7(
.

1. Initialization of the API.

Flashphoner.init()code

For initialization, object Context is passed to the init() method.

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

2. List available media devices.

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. Video render management

FPSurfaceViewRenderer.setMirror()code

When a video is shown, an image is displayed to FPSurfaceViewRenderer objects:

- localRender to display video from camera
- remoteRender to display stream published preview
- newSurfaceRenderer to demonstrate renderer switching

For those objects, screen position, scaling type and mirroring should be set.

By default, mirror view is set to display video from camera by setMirror(true) method invokation. To display stream published preview and renderer switching object, mirroring is switched off by 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();
```

In this case, when you choose front camera, the image displayed from camera looks normally but is published mirror. When you choose back camera, image from camera looks mirror but is publihed in normal orientation (see application screenshots above).

4. Getting audio and video constraints set by user

AudioConstraints, VideoConstraintscode

```
@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().toString()));
        }
    }
    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().toString()));
        }
    }
    return new Constraints(audioConstraints, videoConstraints);
}
```

5. Local camera and microphone testing

Flashphoner.getLocalMediaAccess()code

This parameters are passed:

- audio and video constarints set by user
- local object SurfaceViewRenderer localRenderer to display image from camera

```
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. Session creation

Flashphoner.createSession()code

Object SessionOptionswith the following parameters is passed to thecreateSession() method:

- URL of WCS server
- SurfaceViewRenderer, which will be used to display video from the camera
- SurfaceViewRenderer, which will be used to play the published video stream

```
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. Connection to the server.

Session.connect().code

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

8. Receiving the event confirming 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);
           mTestButton.setEnabled(false);
           mStatusView.setText(connection.getStatus());;
           ...
       }
   });
}
```

9. Video stream creation

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. Video stream publishing.

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.Receiving the event confirming successful stream publishing

StreamStatusEvent PUBLISHINGcode

On receiving this event preview stream is created with Session.createStream() and Stream.play() is invoked to play it.

```
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. Switching camera while publishing stream

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. Switching renderer object while publishing stream

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. Sound volume changing with hardware buttons

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. Device speakerphone usage

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

```
        mSpeakerPhone = (CheckBox) findViewById(R.id.use_speakerphone);
        mSpeakerPhone.setChecked(Flashphoner.getAudioManager().getAudioManager().isSpeakerphoneOn());
        mSpeakerPhone.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
                Flashphoner.getAudioManager().setUseSpeakerPhone(isChecked);
            }
        });
```

16. Session disconnection.

Session.disconnect()code

```
mStartButton.setEnabled(false);

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

17.Receiving the event confirming successful disconnection

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);
      }
   });
}
```