

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

Брянский государственный технический университет

**Утверждаю**

**Ректор университета**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_О.Н.Федонин**

**«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2017 г.**

**ПРОГРАММИРОВАНИЕ МОБИЛЬНЫХ СИСТЕМ**

**Мультимедийные возможности мобильных устройств. Работа с камерой и микрофоном.**

**Методические указания**

**к выполнению лабораторной работы №7**

**для студентов очной формы обучения по направлениям подготовки 02.03.03 – «Математическое обеспечение и администрирование информационных систем», 09.03.01 – «Информатика и вычислительная техника» и 09.03.04 – «Программная инженерия»**

**Брянск 2017**

УКД 004.43

Программирование мобильных систем. Мультимедийные возможности мобильных устройств. Работа с камерой и микрофоном. [Текст] + [Электронный ресурс]: Методические указания к выполнению лабораторной работы №7 для студентов очной формы обучения по направлениям подготовки 02.03.03 – «Математическое обеспечение и администрирование информационных систем», 09.03.01 – «Информатика и вычислительная техника» и 09.03.04 – «Программная инженерия». – Брянск: БГТУ, 2017. – 21с.

Разработал:

Д.Н.Панус

ст.преп.

Рекомендовано кафедрой «Информатика и программное обеспечение» БГТУ (протокол №2 от 16.09.2016)

Научный редактор Д.А.Коростелев

Редактор издательства Л.И.Афонина

Компьютерный набор Д.Н. Панус

Темплан 2017 г., п.273

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подписано в печать 23.09.17. Формат 60х84 1/16 Бумага офсетная. Офсетная печать. Усл.печ.л. 1,8. Уч.-изд.л. 1,8 Тираж 1 экз. Заказ Бесплатно.

Издательство Брянского государственного технического университета

241035, Брянск, бульвар 50-летия Октября, 7, БГТУ. 58-82-49.

Лаборатория оперативной полиграфии БГТУ, ул. Институтская, 16.

# Цель работы

Целью работы является знакомство с методами работы и получения медиа данных с камеры и микрофона устройства. Изучение API голосового управления и распознавания речи.

Продолжительность работы – 5 часов.

# Порядок выполнения работы

1. Произвести выбор выполняемого варианта лабораторной работы на основе своего порядкового номера в группе по алфавиту.
2. Выяснить возможности предоставляемые тестовой средой для демонстрации результата работы(телефон или эмулятор ОС Android).
3. Ознакомиться с теоретическим материалом для лабораторной работы. Разобраться с теорией хранения пользовательской и системной информации приложениями в ОС Android.
4. Проанализировать какие компоненты, используемые в ОС Android необходимо применить в данной лабораторной работе.
5. Произвести проектирование и реализацию интерфейса приложения.
6. Произвести проектирование и реализацию средств хранения информации приложением.
7. Описать в виде XML требуемые ресурсы приложения.
8. Реализовать в приложении общие требования к лабораторной работе.
9. Реализовать специфичные для варианта требования в лабораторной работе, ознакомившись, c правилами их реализации.
10. Произвести сборку готового приложения в виде \*.apk файла приложения.
11. Продемонстрировать полученный результат преподавателю.

# Проигрывание аудио и видео

Android содержит полноценный Медиапроигрыватель, который упрощает проигрывание аудио- и видеоданных.

В Android 2.1 (API level 7) поддержка проигрывания следующих мультимедийных форматов - часть базового фреймворка (имейте в виду, что некоторые устройства могут поддерживать проигрывание дополнительных файловых форматов):

* аудио: AAC LC/LTP, HE-AACv1 (AAC+), HE-AACv2 (Enhanced AAC+), AMR-NB AMR-WB, MP3, MIDI, Ogg Vorbis, PCM / WAVE;
* видео: H.263, H.264 AVC, MPEG-4 SP.

За проигрывание мультимедийных файлов в Android отвечает класс *MediaPlayer*. Вы можете воспроизводить медиаданные, размещенные в ресурсах приложения, локальных файлах, Источниках данных или в сетевом потоке. В объекте *MediaPlayer* управление аудио- и видеофайлами реализовано в виде машины состояний. Если говорить более простым языком, режимы, через которые проходит эта машина состояний, можно описать так:

* инициализация Медиа проигрывателя с помощью заданных мультимедийных данных;
* подготовка объекта MediaPlayer к воспроизведению;
* запуск воспроизведения;
* временная или полная остановка во время воспроизведения;
* завершение воспроизведения. Более детальное и исчерпывающее описание машины состояний Медиа-проигрывателя вы можете найти на сайте для разработчиков по адресу *http:// developer.android.com/reference/android/media/MediaPlayer.html#StateDiagram.*

Чтобы проигрывать мультимедийный ресурс, необходимо создать новый экземпляр класса MediaPlayer, инициализировать его с помощью источника медиаданных и подготовить к воспроизведению. В следующем разделе рассказывается, как нужно инициализировать и подготавливать объект MediaPlayer. После этого вы научитесь управлять процессом воспроизведения, запуская, останавливая и ставя его на паузу, а также перемещаясь по временной шкале. В любом случае, при завершении процесса проигрывания необходимо вызвать метод *release* из объекта MediaPlayer, чтобы освободить соответствующие ресурсы: *mediaPlayer.release();* Android поддерживает ограниченное число одновременно работающих объектов MediaPlayer. Не освобождая их, вы рискуете столкнуться с выбросом исключения, когда система исчерпает ресурсы.

### Инициализация аудиоданных для воспроизведения

Чтобы начать воспроизведение аудиоданных с помощью Медиапро игрывателя, необходимо создать объект MediaPlayer и назначить для него источник с мультимедийной информацией (в нашем случае с аудио). Для воспроизведения аудио с помощью Медиапроигрывателя можно использовать статический метод create, передавая ему в качестве пара- метров Контекст приложения, а также одно из следующих значений:

* идентификатор ресурса;
* путь URI к локальному файлу (используя схему file://);
* путь URI к удаленному ресурсу с аудио, предоставленному в виде URL;
* путь URI к записи внутри локального Источника данных.

Context appContext = getApplicationContext();

MediaPlayer resourcePlayer = MediaPlayer.create(appContext,

R.raw.my\_audio);

MediaPlayer filePlayer = MediaPlayer.create(appContext,

Uri.parse("file:///sdcard/localfile.mp3"));

MediaPlayer urlPlayer = MediaPlayer.create(appContext,

Uri.parse("http://site.com/audio/audio.mp3"));

MediaPlayer contentPlayer = MediaPlayer.create(appContext,

Settings.System.DEFAULT\_RINGTONE\_URI);

В качестве альтернативного можно использовать метод setDataSource из уже созданного экземпляра MediaPlayer. В виде единственного параметра этот метод может принять путь к файлу, путь URI к Источнику данных, адрес URL мультимедийного потока или файловый дескриптор. Используя данный подход, важно не забыть вызвать метод prepare из объекта MediaPlayer.

MediaPlayer mediaPlayer = new MediaPlayer();

mediaPlayer.setDataSource("/sdcard/test.3gp");

mediaPlayer.prepare();

### Подготовка видеоданных к воспроизведению

Воспроизведение видеоданных немного сложнее, чем аудио. Чтобы отобразить видео, необходимо указать поверхность, на которой оно будет выводиться. Наиболее распространенными являются два способа воспроизведения видеоданных. Первый предусматривает использование элемента *VideoView* и инкапсулирует создание и размещение поверхности для отображения видео, а также его подготовку внутри Медиапроигрывателя. Второй позволяет указать собственную поверхность для воспроизведения, управляя исходным экземпляром *MediaPlayer* напрямую.

Самый простой способ проигрывания видео - использование элемента *VideoView*. Он содержит поверхность (объект *Surface*), на которую выводится картинка, а также инкапсулирует все операции по управлению медиапроигрывателем. Как и *MediaPlayer*, *VideoView* поддерживает воспроизведение видео из локальных источников или из потока. Данный элемент скрывает от разработчика инициализацию Медиапроигрывателя, предоставляя удобный API. Чтобы задать видео для воспроизведения, вызывается метод *setVideoPath* или *setVideoUri*. В качестве единственного параметра они принимают путь к локальному файлу, путь URI к Источнику данных или адрес удаленного видеопотока. Завершив инициализацию, можно управлять воспроизведением с помощью методов *start*, *stopPlayback*, *pause* и *seekTo*. *VideoView* также включает метод setKeepScreenOn для предотвращения отключения подсветки экрана во время проигрывания.

VideoView videoView = (VideoView)findViewById(R.id.surface);

videoView.setKeepScreenOn(true);

videoView.setVideoPath("/sdcard/test2.3gp");

if (videoView.canSeekForward())

videoView.seekTo(videoView.getDuration()/2);

videoView.start();

[ . . . какие-либо действия . . . ]

videoView.stopPlayback();

### Подготовка поверхности для воспроизведения видеоданных

Первым делом при использовании Медиапроигрывателя с целью показа видео необходимо подготовить поверхность, на которой это видео будет отображаться. Для этого Медиапроигрывателю с помощью метода *setDisplay* нужно передать объект класса *SurfaceHolder*.

Чтобы добавить объект *SurfaceHolder* в разметку вашего пользовательского интерфейса, используйте элемент *SurfaceView*. *SurfaceView* - обертка вокруг класса *SurfaceHolder*, который в свою очередь служит оберткой класса *Surface*, используемого для обновления изображения из фоновых потоков. Инициализация и передача объекта *SurfaceView* в Медиапроигрыватель показана ниже:

public class MyActivity extends Activity implements SurfaceHolder.

Callback

{

private MediaPlayer mediaPlayer;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

mediaPlayer = new MediaPlayer();

SurfaceView surface = (SurfaceView)findViewById(R.id.surface);

SurfaceHolder holder = surface.getHolder();

holder.addCallback(this);

holder.setType(SurfaceHolder.SURFACE\_TYPE\_PUSH\_BUFFERS);

holder.setFixedSize(400, 300);

}

public void surfaceCreated(SurfaceHolder holder) {

try {

mediaPlayer.setDisplay(holder);

} catch (IllegalArgumentException e) {

Log.d("MEDIA\_PLAYER", e.getMessage());

} catch (IllegalStateException e) {

Log.d("MEDIA\_PLAYER", e.getMessage());

} catch (IOException e) {

Log.d("MEDIA\_PLAYER", e.getMessage());

}

}

public void surfaceDestroyed(SurfaceHolder holder) {

mediaPlayer.release();

}

public void surfaceChanged(SurfaceHolder holder,

int format, int width, int height) { }

}

### Инициализация видеоданных для воспроизведения

После создания и передачи объекта *SurfaceHolder* в Медиапроигрыватель, используется метод *setDataSource*, чтобы указать локальный адрес, URL или путь URI Источника данных, указывающие на ресурс с видео, которое нужно воспроизвести. Выбрав источник с медиаданными, вызовите метод prepare, чтобы инициализировать объект *MediaPlayer* и подготовить его для воспроизведения.

Пример инициализации видеоданных для воспроизведения с помощью объекта *MediaPlayer* представлен ниже.

public void surfaceCreated(SurfaceHolder holder) {

try {

mediaPlayer.setDisplay(holder);

mediaPlayer.setDataSource("/sdcard/test2.3gp");

mediaPlayer.prepare();

mediaPlayer.start();

} catch (IllegalArgumentException e) {

Log.d("MEDIA\_PLAYER", e.getMessage());

} catch (IllegalStateException e) {

Log.d("MEDIA\_PLAYER", e.getMessage());

} catch (IOException e) {

Log.d("MEDIA\_PLAYER", e.getMessage());

}

}

### Управление воспроизведением

Подготовив Медиапроигрыватель, вызовите метод *start*, чтобы начать воспроизведение соответствующих мультимедийных данных: *mediaPlayer.start();*

Используйте методы *stop* и *pause*, чтобы остановить или приостановить проигрывание. Объект *MediaPlayer* также предоставляет методы *getDuration* (позволяет получить длину воспроизводимого аудио или видео) и *getCurrentPosition* (возвращает текущую позицию). Чтобы перейти к определенной позиции, воспользуйтесь методом *seekTo*.

### Управление мультимедийным выводом при воспроизведении

Медиапроигрыватель предоставляет управление громкостью, запрет отключения подсветки экрана во время воспроизведения, а также возможность проигрывать файлы в режиме повторения. В настоящее время нельзя воспроизводить аудиоданные одновременно с разговором по телефону. Медиапроигрыватель всегда использует стандартное устройство для вывода звука - динамик или подключенную по Bluetooth гарнитуру. Используйте методы *isLooping* и *setLooping*, чтобы определить, будет ли файл воспроизводиться в режиме повторения. Чтобы подсветка экрана автоматически не отключалась при воспроизведении, применяйте метод *setScreenOnWhilePlaying*. Он предпочтительнее, чем установка запрета на отключение вручную, поскольку не требует дополнительных полномочий. Во время воспроизведения с помощью метода *setVolume* можно управлять громкостью каждого аудиоканала. В качестве параметров этот метод принимает скалярные значения с плавающей точкой в диапазоне между 0 и 1 для обоих каналов (где 0 - полная тишина, а 1 - максимальная громкость).

# Запись аудио- и видеоданных

Android предлагает два разных пути для записи аудио и видео внутри приложения. Самый простой способ - использование Намерений для запуска стандартного приложения, управляющего камерой. Это позволяет указать качество видео и место, куда его сохранять. Вся работа по записи видео, взаимодействию с пользователем и отлавливанию ошибок «ложится на плечи» стандартного приложения. Если нужен более тонкий контроль над пользовательским интерфейсом или параметрами записи и нужно заменить стандартное приложение, можно воспользоваться классом *MediaRecorder*.

### Использование Намерений в сочетании с объектом MediaRecorder

Самый простой способ инициировать запись видео - использование статической константы *ACTION\_VIDEO\_CAPTURE* из объекта *MediaStore* в качестве параметра для Намерения, которое, в свою очередь, передается методу *startActivityForResult*.

Для записи видео поддерживаются два необязательных параметра, доступных в качестве статических констант класса *MediaStore*.

* **EXTRA\_OUTPUT**. По умолчанию записанное видео сохраняется в источнике MediaStore. Если хотите сохранить его в другом месте, используйте этот параметр для указания альтернативного пути URI.
* **EXTRA\_VIDEO\_QUALITY**. В процессе записи можно указать качество картинки с помощью целочисленного значения. В настоящее время доступны два варианта: 0 - для низкого качества (подходит для MMS) и 1 - для видео с высоким разрешением. По умолчанию выбирается второй вариант.

Ниже показано, как с помощью описанных выше действий записать новый видеофайл с высоким качеством, сохраняя его по указанному пути URI или в стандартный источник *MediaStore*.

private static int RECORD\_VIDEO = 1;

private static int HIGH\_VIDEO\_QUALITY = 1;

private static int MMS\_VIDEO\_QUALITY = 0;

private void recordVideo(Uri outputpath) {

Intent intent = new Intent(MediaStore.ACTION\_VIDEO\_CAPTURE);

if (outputpath != null)

intent.putExtra(MediaStore.EXTRA\_OUTPUT, output);

intent.putExtra(MediaStore.EXTRA\_VIDEO\_QUALITY, HIGH\_VIDEO\_QUALITY);

startActivityForResult(intent, RECORD\_VIDEO);

}

@Override

protected void onActivityResult(int requestCode,

int resultCode, Intent data) {

if (requestCode == RECORD\_VIDEO) {

Uri recordedVideo = data.getData();

// TODO Сделать что-нибудь с записанным видеофайлом

}

}

### Использование объекта MediaRecorder

Запись мультимедийных файлов - прерогатива класса MediaRecorder. С его помощью можно записывать аудио- и видеоданные, которые потом могут быть использованы вашим приложением или сохранены в источник *MediaStore*. Сначала необходимо создать объект *MediaRecorder*. Прежде чем записывать мультимедийные файлы в Android, приложение должно получить полномочия *RECORD\_AUDIO* и/или *RECORD\_VIDEO*. Добавьте в манифест приложения тег *uses-permission* для каждого из этих полномочий.

*MediaRecorder* позволяет указать источник для аудио и видео, формат итогового файла, а также кодировщики, использующиеся для записи. Как и *MediaPlayer*, объект *MediaRecorder* управляет записью с помощью машины состояний. Это значит, что порядок, в котором вы настраиваете запись и управляете ею, играет важную роль.

Иными словами, режимы, через которые проходит эта машина состояний, можно описать так:

* создание нового объекта MediaRecorder;
* назначение источника записи;
* определение итогового формата;
* указание кодировщиков для аудио и видео, частоту кадров и результирующий размер;
* выбор файла, в который будет производиться запись;
* подготовка к записи;
* запись;
* конец записи.

Закончив запись мультимедийного файла, вызовите метод *release* из объекта *MediaRecorder*, чтобы освободить соответствующие ресурсы.

### Запись видео: настройка и управление

Как следует из модели состояний, описанной выше, прежде чем начинать запись, необходимо указать входящий источник, итоговый формат, кодировщики для аудио и видео, а также файл, в который будет вестить запись, - именно в таком порядке.

Методы *setAudioSource* и *setVideoSource* предлагают обозначить одну из статических констант, хранящихся *в MediaRecorder.AudioSource* и *MediaRecorder.VideoSource*, которые определяют источники для аудио- и видеоданных соответственно.

Задав источники для записи, выберите итоговый формат с помощью метода *setOutputFormat*, передав ему одну из констант, хранящихся в *MediaRecorder.OutputFormat*.

Укажите кодировщики для аудио и видео с помощью методов *set[audio/ video]Encoder*, используя константы из класса *MediaRecorder.[Audio/ Video]Encoder*. При желании установите частоту кадров и размер итогового видео.

В завершение с помощью метода *setOutputFile* передайте файл, в который будет производиться запись. И только после этого вызовите метод *prepare*.

### Предварительный просмотр записываемого видео

В процессе записи видео, как правило, не помешает отображать записываемую картинку в режиме реального времени. Используя метод *setPreviewDisplay*, можно задать объект *Surface* для показа видеопотока во время записи.

Работает данный подход так же, как воспроизведение видео с помощью Медиапроигрывателя.. Начните с создания новой Активности, которая включает элемент пользовательского интерфейса *SurfaceView* и реализует интерфейс *SurfaceHolder.Callback*.

Создав объект *SurfaceHolder*, передайте его в метод *setPreviewDisplay*, принадлежащий *MediaRecorder*. Как только вызовется метод *prepare*, на экране в режиме реального времени начнет отображаться записываемый видеопоток.

# Использование камеры и создание снимков

Самый простой способ сделать снимок с помощью камеры - использовать статическую константу *ACTION\_IMAGE\_CAPTURE* из объекта *MediaStore* для создания Намерения, которое потом нужно передать методу *startActivityForResult*.

startActivityForResult(new Intent(MediaStore.ACTION\_IMAGE\_CAPTURE),

TAKE\_PICTURE);

Данный код запустит Активность для управления камерой, позволяя таким образом пользователям вручную изменять настройки изображения.

### Управление камерой и создание снимков

Чтобы получить прямой доступ к аппаратным возможностям камеры, необходимо добавить полномочие CAMERA в манифест своего приложения. Используйте класс *Camera*, чтобы откорректировать настройки камеры, указать параметры изображения и сделать снимок. Чтобы получить доступ к Сервису камеры, применяйте статический метод *open* из класса *Camera*. Когда приложение закончило работу с этим Сервисом, следует освободить его ресурсы с помощью метода *release*. Ниже показан пример использование камеры.

Camera camera = Camera.open();

[ . . . Какие-либо операции с камерой . . . ]

camera.release();

### Отслеживание и изменение настроек камеры и параметров изображения

Настройки камеры хранятся в объекте *Camera.Parameters*, доступ к которому можно получить с помощью метода *getParameters*, вызванного из экземпляра класса *Camera*. Чтобы изменить эти настройки, используйте методы *set\** из объекта *Parameters*, после чего вызовите метод *setParameters*, передав ему модифицированный объект.

В версии Android 2.0 (API level 5) представлены параметры камеры, у каждого из которых свой геттер и сеттер.

* **[get/set]SceneMode**. Принимает или возвращает статическую строковую константу SCENE\_MODE\_\* из класса *Parameters*. Каждый режим описывает определенный тип обстановки («вечеринка», «пляж», «закат» и т. д.).
* **[get/set]FlashMode**. Принимает или возвращает статическую строковую константу FLASH\_MODE\_\*. Позволяет указать режим освещения (включен, выключен или уменьшение эффекта красных глаз) или режим вспышки.
* **[get/set]WhiteBalance**. Принимает или возвращает статическую строковую константу WHITE\_BALANCE\_\*, с помощью которой описывается баланс белого цвета для фотографируемой сцены.
* **[get/set]ColorEffect**. Принимает или возвращает статическую строковую константу EFFECT\_\*, чтобы изменить способ представления изображения. Вам доступны цветовые эффекты, включая тон сепии или оттенки серого.
* **[get/set]FocusMode**. Принимает или возвращает статическую строковую константу FOCUS\_MODE\_\*, которая задает режим автоматической фокусировки камеры.

Параметрами камеры также можно воспользоваться при чтении и записи настроек размера, качества и формата изображения, миниатюры и картинки, отображаемой для предварительного просмотра.

* Качество JPEG и миниатюры. Используйте методы *setJpegQuality* и *setJpegThumbnailQuality*, передавая им целочисленные значения от 0 до 100, где 100 - самое высокое качество.
* Размер изображения, картинки для предварительного просмотра, а также миниатюры. Применяйте методы *setPictureSize*, *setPreviewSize* и *setJpegThumbnailSize*, чтобы задать высоту и ширину изображению, картинке для предварительного просмотра и миниатюре соответственно.
* Растровый формат для изображения и картинки при предварительном просмотре. Используйте методы *setPictureFormat* и *setPreviewFormat*, чтобы задать формат изображения, задействуя статическую константу из класса *PixelFormat*.
* Частота кадров при предварительном просмотре. Применяйте метод *setPreviewFrameRate*, чтобы указать частоту кадров для предварительного просмотра в FPS (количество кадров в секунду).

Каждое конкретное устройство потенциально содержит поддержку собственного подмножества этих значений. Класс *Camera.Parameters* также содержит методы *getSupported\**, с помощью которых можно найти доступные параметры, чтобы потом показать их пользователю. Перед присвоением значения нужно удостовериться, что данный параметр поддерживается. Проверка параметров на допустимость имеет большое значение при выборе доступного режима для предварительного просмотра или размера изображения, так как камеры в различных устройствах могут обладать разными характеристиками.

### Создание фотографий

Сделайте снимок, вызвав метод *takePicture* из объекта *Camera*, передав ему в качестве параметров *ShutterCallback* и две реализации *PictureCallback* (одну для необработанных изображений, другую для сжатых в формат JPEG). Каждая из этих двух функций обратного вызова получит массив байтов - изображение в соответствующем формате, тогда как *ShutterCallback* сработает сразу после закрытия затвора камеры. Ниже показан каркас, в котором создается и сохраняется снимок в формате JPEG на карту SD.

private void takePicture() {

camera.takePicture(shutterCallback, rawCallback, jpegCallback);

}

ShutterCallback shutterCallback = new ShutterCallback() {

public void onShutter() {

// TODO Действия при закрытии затвора камеры.

}

};

PictureCallback rawCallback = new PictureCallback() {

public void onPictureTaken(byte[] data, Camera camera) {

// TODO Действия с необработанным изображением.

}

};

PictureCallback jpegCallback = new PictureCallback() {

public void onPictureTaken(byte[] data, Camera camera) {

// Сохраните изображение в формате JPEG на карту SD

FileOutputStream outStream = null;

try {

outStream = new FileOutputStream("/sdcard/test.jpg");

outStream.write(data);

outStream.close();

} catch (FileNotFoundException e) {

Log.d("CAMERA", e.getMessage());

} catch (IOException e) {

Log.d("CAMERA", e.getMessage());

}

}

};

# Добавление новых мультимедийных данных в MediaStore

По умолчанию мультимедийные файлы, созданные вашим приложением, недоступны для других программ. Выход из этой ситуации - добавление таких файлов в источник *MediaStore*, с помощью которого любое приложение может получить к ним доступ.

Android предоставляет два способа добавления данных в *MediaStore*: использование Сервиса *MediaScannerConnection*, который автоматически распознает и внесет ваш файл, или вручную добавление новой записи в соответствующий Источник данных.

### Использование MediaScannerConnection

Если вы записали медиафайл, класс *MediaScannerConnection* поможет вам легко добавить его в *MediaStore*, избавляя от необходимости вручную создавать запись в этом Источнике данных. Прежде чем использовать метод *scanFile*, чтобы начать сканирование файла, необходимо вызвать метод connect и дождаться установления связи с Сервисом *MediaScannerConnection*. Этот вызов проходит в асинхронном режиме, поэтому вы должны реализовать интерфейс *MediaScannerConnectionClient* - оповестить приложение, что связь была установлена. Можно использовать этот же класс для уведомлений о завершении сканирования, чтобы знать, когда необходимо разорвать связь с *MediaScannerConnection*. Ниже показан каркас для создания нового класса *MediaScannerConnectionClient*, в котором определяется объект *MediaScannerConnection*, необходимый для добавления нового файла в источник *MediaStore*.

MediaScannerConnectionClient mediaScannerClient = new

MediaScannerConnectionClient() {

private MediaScannerConnection msc = null;

{

msc = new MediaScannerConnection(getApplicationContext(), this);

msc.connect();

}

public void onMediaScannerConnected() {

msc.scanFile("/sdcard/test1.jpg", null);

}

public void onScanCompleted(String path, Uri uri) {

msc.disconnect();

}

### Добавление медиафайлов в MediaStore

Вместо того чтобы полагаться на *MediaScannerConnection*, можно самостоятельно добавить медиафайл в *MediaStore*, создав новый объект *ContentValues* и вставив его в соответствующий Источник данных. В качестве метаданных для нового медиафайла нужно указывать название, временную отметку (*timestamp*) и географическую информацию, как показано в следующем фрагменте кода:

ContentValues content = new ContentValues(3);

content.put(Audio.AudioColumns.TITLE, "TheSoundandtheFury");

content.put(Audio.AudioColumns.DATE\_ADDED,

System.currentTimeMillis() / 1000);

content.put(Audio.Media.MIME\_TYPE, "audio/amr");

Также нужно указать абсолютный путь к файлу, который необходимо добавить, получить доступ к объекту *ContentResolver* приложения и использовать его для вставки новой строки в *MediaStore*. Как только медиафайл вставлен в *MediaStore*, необхожимо объявить о его доступности с помощью Широковещательного намерения.

# Работа с несжатым звуком

Классы *AudioTrack* и *AudioRecord* позволяют напрямую записывать аудио и воспроизводить аудиопоток в формате PCM, используя аппаратные возможности устройства.

С помощью *AudioTrack* можно обрабатывать входящий аудиопоток и воспроизводить его практически в режиме реального времени, что дает возможность манипулировать входящим и исходящим звуком, обрабатывая на устройстве несжатые аудиоданные.

### Запись звука с помощью AudioRecord

Используйте класс *AudioRecord* для записи аудиоданных непосредственно с аппаратных буферов. Создайте новый объект *AudioRecord*, указав источник, частоту, настройки каналов, кодировщик для аудио и размер буфера.

int bufferSize = AudioRecord.getMinBufferSize(frequency,

channelConfiguration,

audioEncoding);

AudioRecord audioRecord = new AudioRecord(MediaRecorder.AudioSource.MIC,

frequency,

channelConfiguration,

audioEncoding, bufferSize);

Из соображений конфиденциальности Android требует, чтобы в манифесте приложения значилось полномочие RECORD\_AUDIO.

Значения частоты, кодировщика аудио и настроек каналов повлияют на размер и качество записанного аудиофайла. Обратите внимание, что ни один из этих параметров не связан с самими файлами. Когда объект *AudioRecord* инициализирован, вызовите метод *startRecording*, чтобы начать асинхронную запись. Используйте метод *read* для добавления необработанных аудиоданных в записываемый буфер.

Ниже демонстрируется запись несжатого аудио с микрофона с последующим сохранением в файл, размещенный на карте SD.

int frequency = 11025;

int channelConfiguration = AudioFormat.CHANNEL\_CONFIGURATION\_MONO;

int audioEncoding = AudioFormat.ENCODING\_PCM\_16BIT;

File file = new File(Environment.getExternalStorageDirectory(), "raw.pcm");

// Создайте новый файл.

try {

file.createNewFile();

} catch (IOException e) {}

try {

OutputStream os = new FileOutputStream(file);

BufferedOutputStream bos = new BufferedOutputStream(os);

DataOutputStream dos = new DataOutputStream(bos);

int bufferSize = AudioRecord.getMinBufferSize(frequency,

channelConfiguration,

audioEncoding);

short[] buffer = new short[bufferSize];

// Создайте новый объект AudioRecord, чтобы записать звук.

AudioRecord audioRecord = new AudioRecord(MediaRecorder.AudioSource.MIC,

frequency,

channelConfiguration,

audioEncoding, bufferSize);

audioRecord.startRecording();

while (isRecording) {

int bufferReadResult = audioRecord.read(buffer, 0, bufferSize);

for (int i = 0; i < bufferReadResult; i++)

dos.writeShort(buffer[i]);

}

audioRecord.stop();

dos.close();

} catch (Throwable t) {}

### Воспроизведение звука с помощью AudioTrack

Используйте класс *AudioTrack*, чтобы воспроизводить звук напрямую через аппаратные буферы устройства. Создайте новый объект *AudioTrack*, указав потоковый режим, частоту, параметры каналов, тип кодировщика и длину аудио. Поскольку аудиоданные необработанные, отсутствует метаинформация, связанная с ними. Поэтому важно установить корректные свойства, чтобы они совпадали с теми, которые были использованы при записи файла. Инициализировав объект *AudioTrack*, вызовите метод *play*, чтобы начать асинхронное воспроизведение. Используйте метод *write*, чтобы добавить «сырые» аудиоданные в буфер проигрывателя. Вы можете начать запись в буфер объекта *AudioTrack* как до вызова метода *play*, так и после. В первом случае воспроизведение пойдет сразу после вызова, во втором - звук станет проигрываться, как только запишете данные в буфер *AudioTrack*.

Ниже показано воспроизведение файла, записанного ранее, при этом ожидаемая частота аудиофайла удваивается, что приводит к увеличению скорости проигрывания вдвое.

int frequency = 11025/2;

int channelConfiguration = AudioFormat.CHANNEL\_CONFIGURATION\_MONO;

int audioEncoding = AudioFormat.ENCODING\_PCM\_16BIT;

File file = new File(Environment.getExternalStorageDirectory(), "raw.pcm");

// Массив типа short для хранения аудиоданных (звук 16-битный,

// поэтому выделяем по 2 байта на значение)

int audioLength = (int)(file.length()/2);

short[] audio = new short[audioLength];

try {

InputStream is = new FileInputStream(file);

BufferedInputStream bis = new BufferedInputStream(is);

DataInputStream dis = new DataInputStream(bis);

int i = 0;

while (dis.available() > 0) {

audio[audioLength] = dis.readShort();

i++;

}

// Закрытие входящих потоков.

dis.close();

// Создание объекта AudioTrack и проигрывание звука с его помощью

AudioTrack audioTrack = new AudioTrack(AudioManager.STREAM\_MUSIC,

frequency,

channelConfiguration,

audioEncoding,

audioLength,

AudioTrack.MODE\_STREAM);

audioTrack.play();

audioTrack.write(audio, 0, audioLength);

} catch (Throwable t) {}

# Распознавание речи

Начиная с версии 1.5 (API level 3) Android поддерживает голосовой ввод и распознавание речи с помощью класса *RecognizerIntent*. Этот API позволяет встраивать в приложения голосовой ввод, используя стандартное диалоговое окно.

Речь распознается с помощью метода *startNewActivityForResult*, которому необходимо передать Намерение, созданное с применением константы *RecognizerIntent.ACTION\_RECOGNIZE\_SPEECH*. Намерение также должно содержать дополнительный параметр *RecognizerIntent.EXTRA\_LANGUAGE\_MODEL*, указывающий на языковую модель распознавания входящего звука. Он может содержать следующие значения: *LANGUAGE\_MODEL\_FREE\_FORM* или *LANGUAGE\_MODEL\_WEB\_SEARCH*. Оба доступны в виде статических констант из класса *RecognizerIntent*. Еще вы можете указать несколько необязательных параметров, чтобы изменять язык, количество потенциальных результатов и содержание отображаемой строки, используя следующие константы из класса *RecognizerIntent*:

* **EXTRA\_PROMPT** - задайте строку, которая будет отображаться в окне голосового ввода). Она должна предлагать пользователю сказать что-нибудь в микрофон;
* **EXTRA\_MAXRESULTS** - используйте целочисленное значение, чтобы ограничить количество результатов, которые могут быть возвращены при распознавании;
* **EXTRA\_LANGUAGE** - укажите языковую константу из класса Locale, чтобы задать язык для ввода, отличный от языка по умолчанию на данном устройстве. Вы можете получить текущее значение по умолчанию, вызвав статический метод *getDefault* из класса *Locale*.

Ниже показывается, как инициировать распознавание речи на английском языке, используя собственную строку в диалоговом окне и ограничивая количество результатов до одного.

Intent intent = new Intent(RecognizerIntent.ACTION\_RECOGNIZE\_SPEECH)

// Укажите свободную форму ввода

intent.putExtra(RecognizerIntent.EXTRA\_LANGUAGE\_MODEL,

RecognizerIntent.LANGUAGE\_MODEL\_FREE\_FORM);

intent.putExtra(RecognizerIntent.EXTRA\_PROMPT,

"or forever hold your peace");

intent.putExtra(RecognizerIntent.EXTRA\_MAX\_RESULTS, 1);

intent.putExtra(RecognizerIntent.EXTRA\_LANGUAGE, Locale.ENGLISH);

startActivityForResult(intent, VOICE\_RECOGNITION);

Когда пользователь завершит голосовой ввод, записанный звук проанализируется и обработается движком для распознавания речи. Результаты вернутся через обработчик *onActivityResult* в виде списка строк, хранящегося в параметре EXTRA\_RESULTS. Ниже показывается извлечение результатов распознавания речи.

@Override

protected void onActivityResult(int requestCode,

int resultCode,

Intent data) {

if (requestCode == VOICE VOICE\_RECOGNITION && resultCode == RESULT\_OK)

{

ArrayList<String> results;

results = data.getStringArrayListExtra(RecognizerIntent.EXTRA\_

RESULTS);

// TODO Выполнить какие-то действия с распознанными строками

}

super.onActivityResult(requestCode, resultCode, data);}

# Задание на лабораторную работу

Необходимо создать Android приложение, способное получать информацию с доступных на устройстве мультимедийных сенсоров: камеры и микрофона.

**Общие требования** для выполнения лабораторной работы:

1. Реализовать работу с камерой или микрофоном. Конкретное задание определяется вариантом задания.
2. Дополнительной реакцией на нажатие кнопок на экране должна быть вибрация.

**Типы заданий**:

1. Реализовать отображение информации с камеры на SurfaceView объекте, и запись видео, получаемого в этот момент с камеры.
2. Реализовать отображение видео данных с использованием SurfaceView.
3. Реализовать отображение видео данных с использованием стандартных инструментов и приложений.
4. Реализовать запись звука с микрофона устройства.
5. Реализовать воспроизведение звука стандартными средствами.
6. Реализовать воспроизведение звука с иcпользованием Android API (Низкий уровень ).
7. Реализовать функционал приложения: фотокамера, которая отображает получаемое с камеры изображение на surfaceView объекте и сохраняет изображение по требованию пользователя.
8. Реализовать экран голосового ввода данных, с использованием Andpoid API.

Комбинации заданий по вариантам:

1) 1,3

2) 4,6

3) 4,5

4) 3,5

5) 1,4

6) 2,5

7) 1,5

8) 3,8

9) 3,6

10) 2,4

11) 1,8

12) 3,6,8

13) 2,4

Итог лабораторной работы - \*.apk файл приложения, способный работать на устройстве с Android (эмулятор, при отсутствии возможности работы с физическим устройством) с версией API от 15 до 18, с целевой версией API 15 (Android 4.0.3)

**Контрольные вопросы**

1. Какие форматы данных Android может воспроизводить стандартными средствами?
2. Вызов системных приложений записи звука и видео.
3. Класс MediaPlayer. Его назначение и особенности.
4. Воспроизведение данных средствами VideoView.
5. SurfaceView при работе с видеопотоком. Назначение и особенности.
6. Управление воспроизведением. Состояния проигрывателя при воспроизведении.
7. Запись звука.
8. Объект MediaRecorder назначение и возможности.
9. Настройка объекта MediaRecorder
10. Какие разрешения необходимо запросить у пользователя для записи звука, видео, или получения изображений с камеры?
11. Просмотр Видеопотока на экране. Класс SurfaceHolder.
12. Для чего необходимо реализовывать 2 основных callback функциии у SurfaceHolder. Что это за функции?
13. Создание снимков камерой. Uses-permission и настройки.
14. Класс AudioRecord. Особенности.
15. Воспроизведение звука с помощью класса AudioTrack.
16. Распознавание речи. Настройки и особенности обработки результатов.

# Список рекомендуемой литературы

1. Майер, Р. Android 2: Программирование приложений для планшетных компьютеров и смартфонов:[пер. с англ] / Р. Майер. - М.: Эксмо, 2011. - 672с.
2. Коматинени, С. Android 4 для профессионалов. Создание приложений для планшетных компьютеров и смартфонов:[пер. с англ]/ С. Коматинени, Д. Маклин. - М.:2012. 880с.
3. Медникс, З. Программирование под Android:[пер. с англ]/ З. Медникс, Л. Дорнин, Б. Мик, М. Накамура. - Спб.:Питер, 2012. - 496с.
4. Аndroid Developers. режим доступа: http://developer.android.com