

Использование сервисов смартфона в мобильных приложениях

Рассмотрим наиболее интересные возможности смартфонов, которые можно использовать в приложениях. С одной стороны, *смартфон* выполняет все привычные функции мобильного телефона и, благодаря компактным размерам, всегда под рукой. С другой стороны, благодаря наличию процессора и операционной системы, позволяет выполнять многие функции полноценного компьютера. Дополнительно ко всему, смартфоны обладают рядом интересных особенностей, не характерных для телефонов и компьютеров.

Для начала обратим внимание на экран смартфона. В современных смартфонах экран занимает практически всю *площадь* передней панели устройства, имеет высокое разрешение и является чувствительным к прикосновениям. Благодаря такой чувствительности, для взаимодействия с устройством и его приложениями можно использовать виртуальные *элементы управления*, чаще всего кнопки, отображаемые на экране. В связи, с чем отпадает необходимость в физических кнопках. В смартфонах реализуется, так называемый, *touch-интерфейс*, основанный на виртуальных элементах управления, выбор которых выполняется простым касанием, а также на использовании жестов (*gestures*).

Если точек касания несколько (т.е. используется несколько пальцев), такой *интерфейс* называется *multi-touch*.

Смартфоны можно использовать в качестве аудио или видеоплеера. В состав платформы *Android* входит набор библиотек для обработки *мультимедиа Media Framework*, в котором реализована *поддержка* большинства общих медиа-форматов. В связи с чем, в приложения, разрабатываемые для смартфонов под управлением *Android*, можно интегрировать *запись* и воспроизведение аудио и видео, а также работу с изображениями.

Важной и часто используемой особенностью смартфонов является наличие камеры. С ростом возможностей получения фото и видео материалов увеличивается потребность в приложениях, способных работать с этими материалами. Платформа *Android* позволяет разрабатывать такие приложения, которые предоставляют пользователям возможности делать фотоснимки или записывать видео, каким-то образом обрабатывать полученные материалы и использовать их далее.

Большинство смартфонов оснащены *GPS*-модулем, а некоторые даже комбинированным модулем *GPS/ГЛОНАСС*, что позволяет использовать такое устройство в качестве инструмента для ориентирования на местности. Во многих случаях *смартфон* с установленным соответствующим программным обеспечением вполне может заменить *GPS* навигатор.

В разрабатываемых приложениях иногда бывает очень полезно добавить возможность получения координат устройства и хозяина, если оба находятся в одном месте, и использовать эти *координаты* для каких-либо целей.

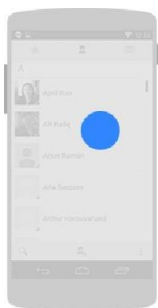
Например, уже существуют приложения, которые позволяют отслеживать параметры человека (спортсмена) во время преодоления некоторых расстояний бегом, на велосипеде, на лыжах и т. д. Такое *приложение* работает во время тренировки (устройство должно перемещаться вместе со спортсменом), по окончании можно получить полную статистику маршрута: *точное время* в пути, *расстояние*, подъемы/спуски, среднюю скорость, потраченные калории и т. д.

Заметим, что большая часть информации опирается на данные, полученные со спутников *GPS*.

Рассмотрение особенностей смартфонов будет неполным, если оставить без внимания датчики и сенсоры, которыми оснащены большинство устройств. Эти микроустройства обеспечивают связь смартфона с окружающей средой и добавляют новые удивительные функции. С помощью датчика приближения, например, можно отключать подсветку экрана при приближении телефона к уху пользователя во время разговора, блокировать экран, чтобы не было возможности случайно нажать на отбой. *Акселерометр* может использоваться для смены ориентации экрана, для управления в играх, особенно симуляторах, а также в качестве шагомера. Датчик освещенности позволяет регулировать яркость экрана. *Гироскоп* может применяться для определения более точного позиционирования устройства в пространстве.

Сенсорное (touch) управление

Далее рассмотрим возможности добавления сенсорного управления в мобильные приложения под *Android*. Сенсорное управление подразумевает использование сенсорных жестов для взаимодействия с приложением. Ниже представлен набор жестов, поддерживаемый системой *Android*.



Касание (touch).

Использование: Запуск действия по умолчанию для выбранного элемента.

Выполнение: нажать, отпустить.



Длинное касание (long touch). **Использование:** Выбор элемента. Не стоит использовать этот жест для вызова контекстного меню.

Выполнение: нажать, ждать, отпустить.



Скольжение или перетаскивание (swipe or drag).

Использование: Прокрутка содержимого или навигация между элементами интерфейса одного уровня иерархии.

Выполнение: нажать, переместить, отпустить.



Скольжение после длинного касания (long press drag).

Использование: Перегруппировка данных или перемещение в контейнер.

Выполнение: длительное касание, переместить, отпустить.



Двойное касание (double touch). **Использование:** Увеличение масштаба, выделение текста.

Выполнение: быстрая последовательность двух касаний.



Перетаскивание с двойным касанием (double touch drag).

Использование: Изменение размеров: расширение или сужение по отношению к центру жеста. **Выполнение:** касание, следующее за двойным касанием со смещением вверх или вниз при этом:

смещение вверх уменьшает размер содержимого;

смещение вниз увеличивает размер содержимого.



Сведение пальцев (pinch close). **Использование:** уменьшение содержимого, сворачивание.

Выполнение: касание экрана двумя пальцами, свести, отпустить.



Разведение пальцев (pinch open). **Использование:** увеличение содержимого, разворачивание.

Выполнение: касание экрана двумя пальцами, развести, отпустить.

О возможности управлять приложением с помощью сенсорных жестов можно говорить в том случае, когда *приложение* способно распознать, что под набором касаний экрана скрывается некоторый жест и выполнить соответствующее действие. Процесс распознавания жеста обычно состоит из двух этапов: сбор данных и *распознавание* жеста.

Сбор данных о сенсорных событиях

Основные действия, которые может произвести пользователь при взаимодействии с сенсорным экраном: коснуться экрана пальцем, переместить палец по экрану и отпустить. Эти действия распознаются системой Android, как сенсорные события (touch-события).

Каждый раз при появлении сенсорного события инициируется вызов метода `onTouchEvent()`. Обработка события станет возможной, если этот метод реализован в классе *активности* или некоторого компонента, иначе событие просто игнорируется.

Жест начинается, при первом касании экрана, продолжается пока система отслеживает положение пальцев пользователя и заканчивается получением финального события, состоящего в том, что ни один палец не касается экрана. Объект `MotionEvent`, передаваемый в метод `onTouchEvent()`, предоставляет детали каждого взаимодействия. Рассмотрим основные константы класса `MotionEvent`, определяющие сенсорные события:

- `MotionEvent.ACTION_DOWN` касание экрана пальцем, является начальной точкой для любого сенсорного события или жеста;
- `MotionEvent.ACTION_MOVE` перемещение пальца по экрану;
- `MotionEvent.ACTION_UP` поднятие пальца от экрана.

Приложение может использовать предоставленные данные для распознавания жеста.

Можно реализовать свою собственную обработку событий для распознавания жеста, таким образом можно работать с произвольными жестами в приложении. Если же в приложении необходимо использовать стандартные жесты, описанные выше, можно воспользоваться классом `GestureDetector`. Этот класс позволяет распознать стандартные жесты без обработки отдельных сенсорных событий.

Распознавание жестов

Android предоставляет класс `GestureDetector` для распознавания стандартных жестов. Некоторые жесты, которые он поддерживает включают: `onDown()`, `onLongPress()`, `onFling()` и т. д. Можно использовать класс `GestureDetector` в связке с методом `onTouchEvent()`. Подробно распознавание поддерживаемых жестов рассмотрено в первой части лабораторной работы в этой теме.

Начиная с версии 1.6, Android предоставляет API для работы с жестами, который располагается в пакете `android.gesture` и позволяет сохранять, загружать, создавать и распознавать жесты. Виртуальное устройство Android (AVD), начиная все с той же версии 1.6, содержит предустановленное приложение, которое называется `Gesture Builder` и позволяет создавать жесты. После создания жесты сохраняются на SD карте виртуального устройства и могут быть добавлены в приложение в виде бинарного ресурса.

Для распознавания жестов необходимо добавить компонент `GestureOverlayView`, в XML файл *активности*. Этот компонент может быть добавлен как обычный элемент графического интерфейса пользователя и встроен в компоновку, например `RelativeLayout`. С другой стороны он может быть использован, как прозрачный слой поверх других компонентов, в этом случае в XML файле активности он должен быть записан, как корневой элемент.

Кроме всего вышеперечисленного, для использования собственных жестов в приложении необходимо реализовать интерфейс `OnGesturePerformedListener` и его метод `onGesturePerformed()`. Подробно создание и использование собственных жестов рассмотрено во второй части лабораторной работы в этой теме.

Работа с мультимедиа

Мультимедиа библиотека *Android* включает поддержку воспроизведения множества наиболее распространенных форматов, что позволяет легко использовать в приложениях аудио, видео и изображения. Можно проигрывать аудио или видео из медиафайлов сохраненных как ресурсы приложения, из файлов, расположенных в файловой системе или из потока данных, получаемого через сетевое соединение, для всего этого используется *MediaPlayer API*.

Замечание: проигрывать аудиофайлы можно только на стандартном устройстве вывода, невозможно воспроизводить аудио во время звонка.

Для воспроизведения аудио и видео *Android* предоставляет класс *MediaPlayer*. Причем, при работе с аудиоконтентом этот класс позволяет воспроизводить необработанные данные, т.е. возможно проигрывание динамически генерируемого аудио.

Диаграмма жизненного цикла экземпляра класса *MediaPlayer* представлена на рисунке 1.

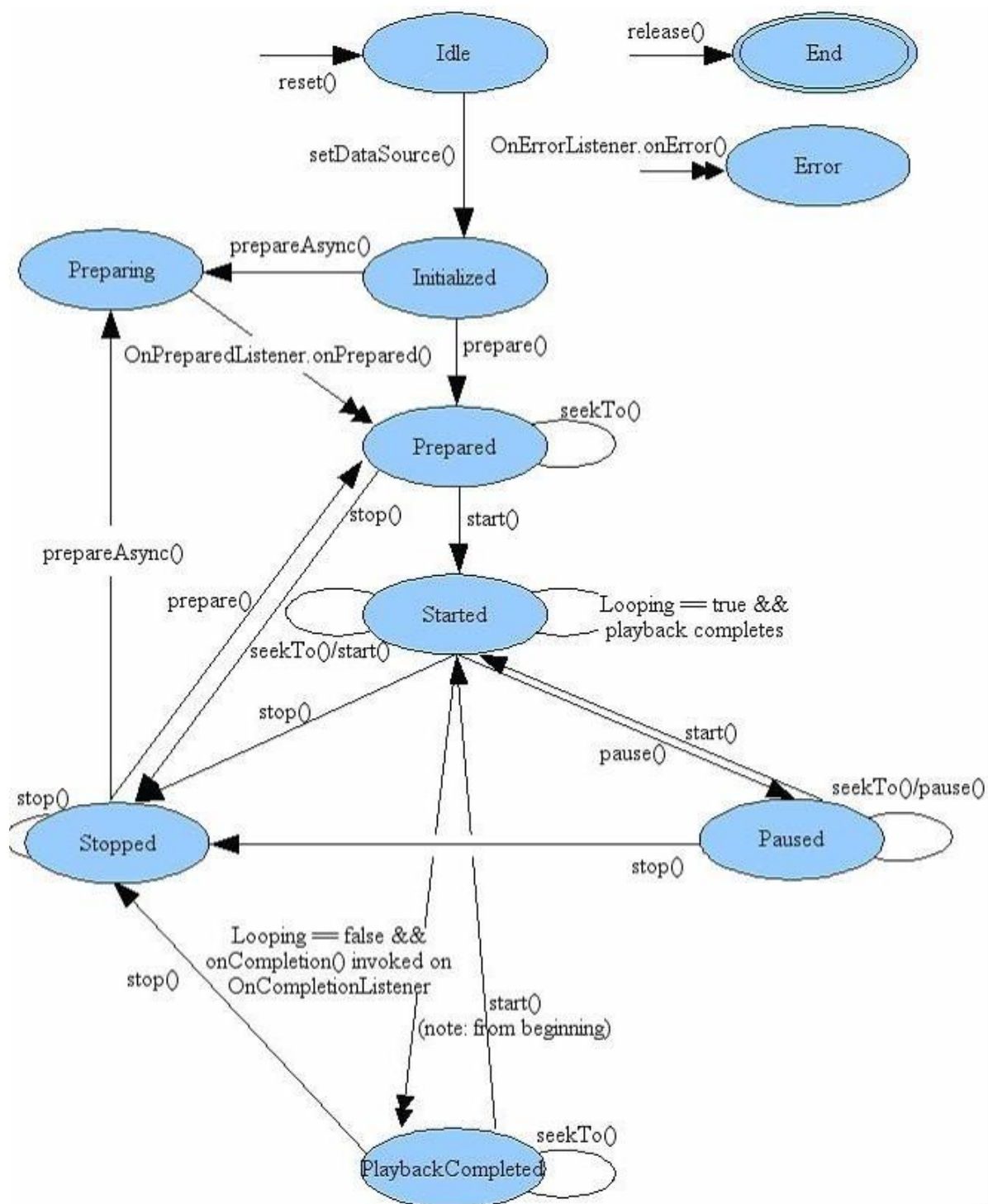


Рисунок 1. Жизненный цикл экземпляра класса *MediaPlayer*

Овалы представляют состояния объекта *MediaPlayer*, дуги показывают вызовы каких методов необходимо выполнить, чтобы сменить состояние объекта *MediaPlayer*. Дуги

с одной стрелкой представляют вызовы синхронных методов, с двумя стрелками вызовы асинхронных методов.

В ходе жизненного цикла объект MediaPlayer проходит через несколько состояний:

- **бездействие (Idle)** создан экземпляр класса MediaPlayer для создания может использоваться оператор new или вызов метода reset();
- **инициализирован (Initialized)** задан источник медиаинформации, для задания источника используется метод setDataSource();
- **ошибка (Error)** появилась какая-то ошибка, например, не поддерживаемый аудио/видео формат, слишком высокое разрешение, чтобы вывести объект из этого состояния, необходимо вызвать метод reset();
- **подготовка (Preparing)** MediaPlayer занимается подготовкой медиаисточника к воспроизведению, подготовка иницируется методом prepareAsync();
- **готов (Prepared)** состояние готовности к воспроизведению, может быть достигнуто двумя способами:

– синхронный способ: вызов метода prepare(), который переводит объект в готовое состояние;

– асинхронный способ: срабатывание метода onPrepared() интерфейса OnPreparedListener() в состоянии подготовки, как реакция на событие готовности;

- **запущен (Started)** выполняется воспроизведение медиа-контента, в это состояние объект переходит после вызова метода start();
- **приостановлен (Paused)** воспроизведение приостановлено, MediaPlayer переходит в это состояние после вызова метода pause();
- **остановлен (Stopped)** воспроизведение остановлено, MediaPlayer переходит в это состояние после вызова метода stop();
- **воспроизведение завершено (Playback Completed)** достигнут конец воспроизводимого содержания, в это состояние объект переходит после срабатывания метода onCompleted() интерфейса слушателя OnCompletionListener, как реакции на конец воспроизводимого материала;

Замечание: из состояний Paused, Playback Completed можно вернуться к воспроизведению вызовом метода start(). Из состояния Stopped прежде, чем вернуться в состояние воспроизведения, необходимо пройти через подготовку медиа-содержимого.

Вызов метода seekTo() позволяет поменять место воспроизведения.

- **конец (End)** конец жизненного цикла MediaPlayer объекта, в это состояние объект переходит после вызова метода release().

Для записи аудио и видео *Android* предоставляет класс MediaRecorder. *Диаграмма* жизненного цикла экземпляра класса MediaRecorder представлена на рисунке 2. Овалы представляют состояния объекта MediaRecorder, дуги показывают вызовы каких методов необходимо выполнить, чтобы сменить состояние объекта MediaRecorder.

Дуги с одной стрелкой представляют вызовы синхронных методов, с двумя стрелками вызовы асинхронных методов.

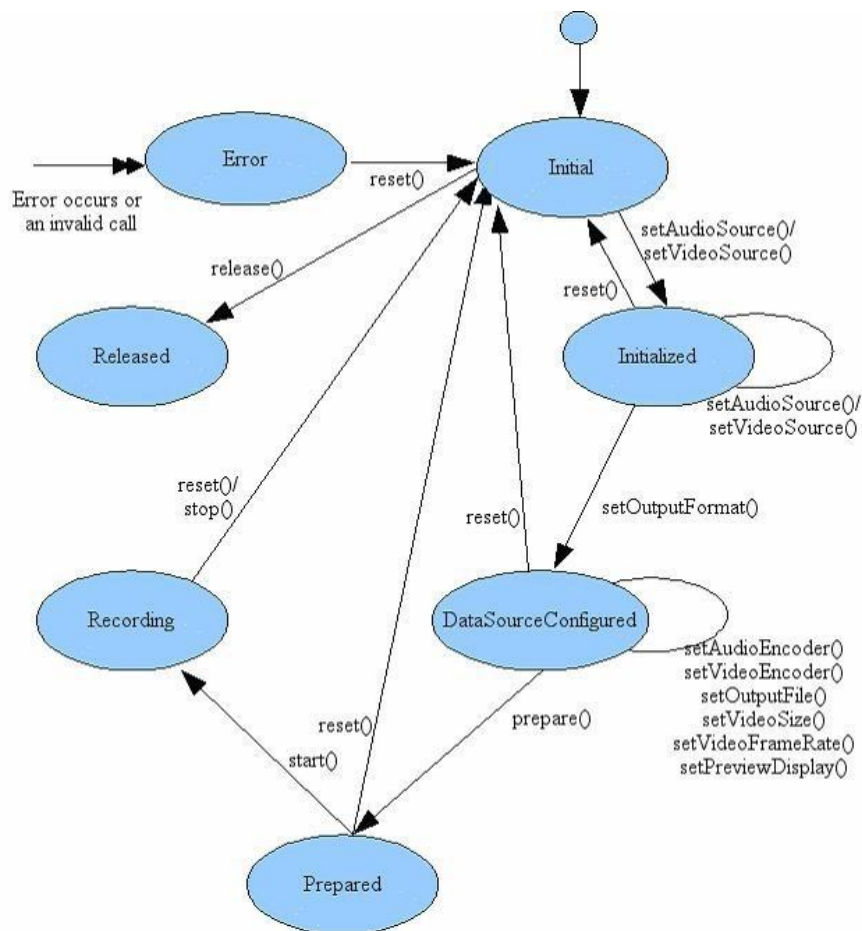


Рисунок 2. Жизненный цикл экземпляра класса MediaRecorder

В ходе жизненного цикла объект MediaRecorder проходит через несколько состояний:

начальное (Initial)	создан объект класса MediaRecorder, для создания может использоваться оператор new или вызов метода reset()
инициализирован (Initialized)	объект MediaRecorder готов к использованию, в данное состояние объект переходит после вызова одного из методов setAudioSource() или setVideoSource(), которые задают источники аудио или видео для записи;
сконфигурирован приемник данных для записи (Data Source Configured)	задаются основные свойства приемника данных, состояние иницируется методом setOutputFormat(), для настройки свойств должны быть выполнены некоторые методы из списка: setAudioEncoder(), setVideoEncoder(), setOutputFile(), setVideoSize(), setVideoFrameRate(), setPreviewDisplay();
готов (Prepared)	состояние готовности к записи, иницируется методом prepare();
записывает (Recording)	идет запись, иницируется вызовом метода start();

освобожден (Released)	запись завершена, все ресурсы освобождены.
-----------------------	--

Использование встроенной камеры

Платформа Android включает поддержку камеры, доступной на устройстве, позволяющей приложениям получать фотографии и записывать видео. Для решения этих задач, существует два способа:

1. непосредственное обращение к камере;
2. использование намерений (Intent) для вызова существующего приложения.

Рассмотрим основные относящиеся к делу классы:

Camera – класс, реализующий управление камерами устройства. Этот класс используется для получения фотографий или записи видео при создании приложения, работающего с камерой.

SurfaceView – класс, используемый для предоставления пользователю возможности предварительного просмотра.

MediaRecorder – класс, используемый для записи видео с камеры.

Intent – класс, содержащий абстрактное описание выполняемой операции, которое передается системе Android, а ОС сама находит и запускает необходимое приложение и возвращает результат его работы.

Для работы с камерой используются два типа намерений:

MediaStore.ACTION_IMAGE_CAPTURE – для запроса на выполнение фотоснимков;

MediaStore.ACTION_VIDEO_CAPTURE – для запроса на запись видео.