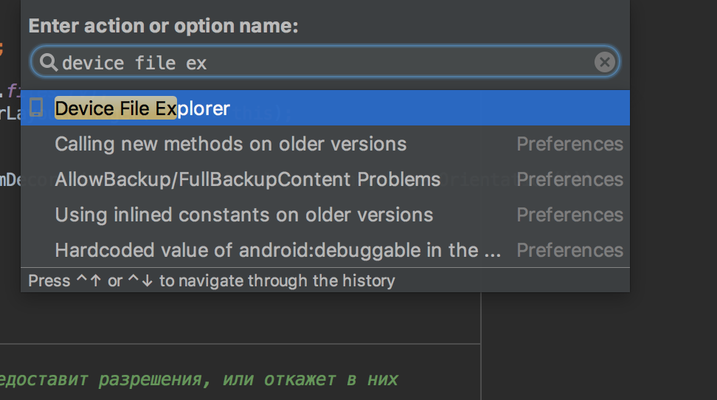
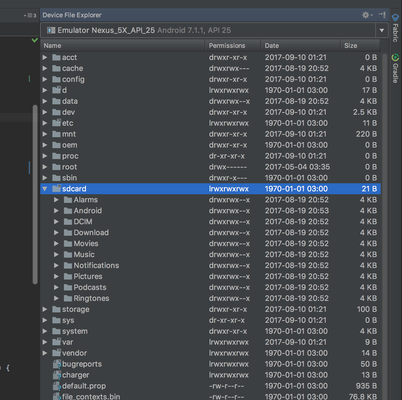
Прежде чем мы приступим к программированию, давайте сделаем следующую вещь: возьмите любой APK-файл, и положите его на SD-карту устройства.

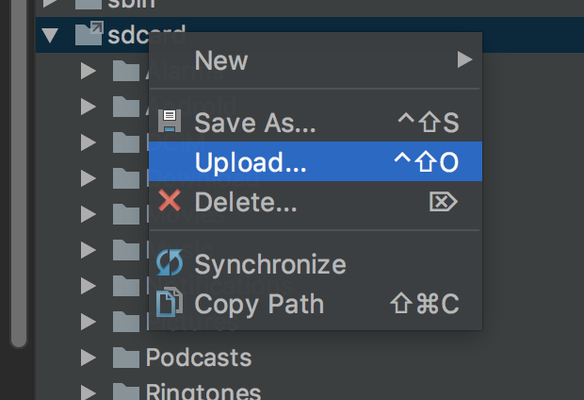
Это можно достаточно легко сделать, используя Android Studio. Нажмите Ctrl + Shift + A (Cmd + Shift + Aна macOS), в появившемся окне введите "Device File Explorer", выберите соответствующий пункт с помощью стрелок **вверх** или **вниз**, и нажмите Enter:

[](https://skillberg.com/media/uploads/2017/11/17/4486e6dba54c4017a247f76d878c82cb.png)Окно выбора действия

Появится вот такое окно:

[](https://skillberg.com/media/uploads/2017/11/17/1240d9519dd54d4fb038a8809848073b.png)Device File Explorer

В нём выберите директорию /sdcard/, нажмите на неё правой кнопкой и выберите **Upload**:

[](https://skillberg.com/media/uploads/2017/11/17/9c0ebdda114d482085c27ddd93085260.png)Загрузка файла

Далее просто выберите APK-файл, и Android Studio загрузит его на виртуальную SD-карту устройства.

Различные типы данных в RecyclerView

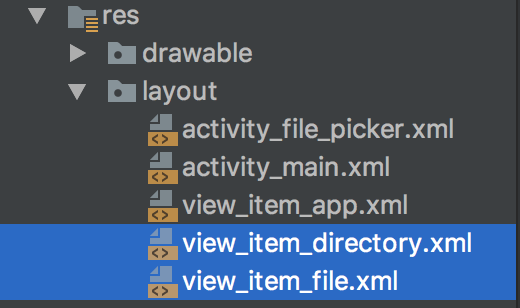
Менеджер файлов, как ни странно, отображает файлы. А файлы (внезапно!) могут оказаться директориями. Было бы неплохо уметь отличать директории от файлов, причем как внешне, так и, например, при обработке кликов на элемент списка.

Благо, разработчики Android позаботились о нас и добавили возможность отличать различные типы данных в RecyclerView.

Давайте договоримся, чем будут отличаться директории и файлы внешне. Для наглядности мы создадим ещё один лэйаут, который будет отображать директории (лэйаут для файлов у нас уже есть), и отличаться они будут только иконками.

Это не совсем практично, ведь можно было обойтись и одним файлом, но в качестве демо отлично подойдёт.

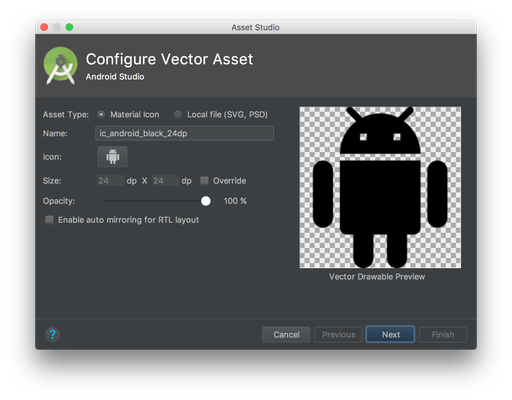
Итак, просто скопируйте лэйаут файла и вставьте его в ту же директорию res/layout, но с именем view\_item\_directory.xml:

[](https://skillberg.com/media/uploads/2017/11/17/06ec2681ebba4bf8b77de51cfa61670f.png)Файл вёрстки ячейки директории

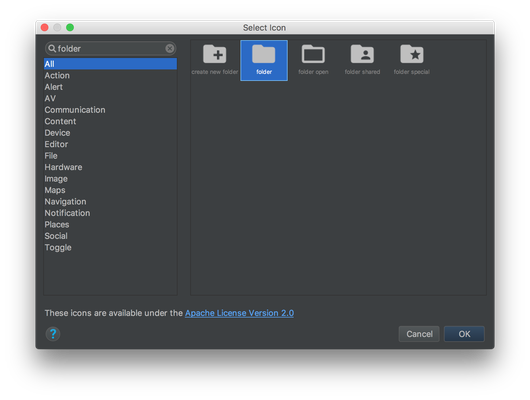
Vector Asset

Теперь нужно добавить иконки директорий и файлов. Вообще, можно найти их в интернете, но ради разнообразия попробуем создать векторные иконки, благо, делается это очень просто.

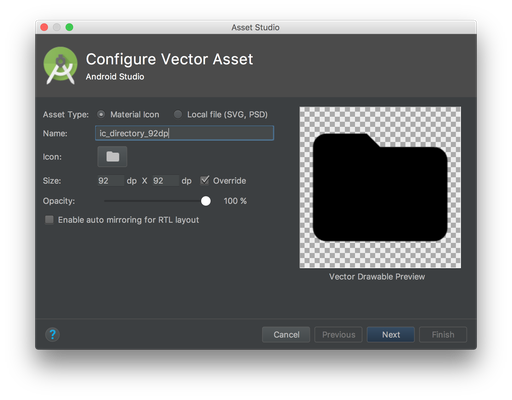
Нажмите правой кнопкой на директорию res/, затем выберите **New** -> **Vector Asset**:

[](https://skillberg.com/media/uploads/2017/11/17/721cb147f0184f10adf00161377f6864.png)Создание векторного изображения

В качестве имени введите ic\_directory\_92dp, затем нажмите на кнопку **Icon**, в появившемся окне введите "folder" и выберите иконку:

[](https://skillberg.com/media/uploads/2017/11/17/36e67727fbc04831b2cd1215cff0be4d.png)Выбор иконки для векторного изображения

Напротив **Size** поставьте галочку **Override** и измените размеры на 92x92:

[](https://skillberg.com/media/uploads/2017/11/17/db663150c8e346ab9dd08c63880a0dae.png)Размеры векторного изображения

Нажмите **Next**, а затем — **Finish**. В директории res/drawable создался файл с векторным изображением:

<vector xmlns:android="http://schemas.android.com/apk/res/android"

android:width="92dp"

android:height="92dp"

android:viewportHeight="24.0"

android:viewportWidth="24.0">

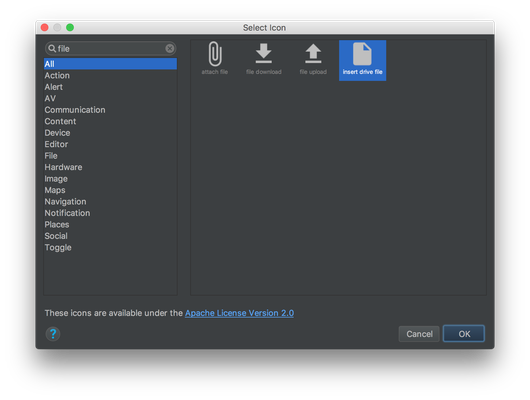
<path

android:fillColor="#FF000000"

android:pathData="M10,4H4c-1.1,0 -1.99,0.9 -1.99,2L2,18c0,1.1 0.9,2 2,2h16c1.1,0 2,-0.9 2,-2V8c0,-1.1 -0.9,-2 -2,-2h-8l-2,-2z" />

</vector>

То же самое проделываем с файлом, только выбираем другую иконку и называем его ic\_file\_92dp:

[](https://skillberg.com/media/uploads/2017/11/17/03980e3cde4f4e80921057b3303f89eb.png)Выбор иконки для другого векторного изображения

Теперь открываем view\_item\_file.xml и вносим следующие изменения в ImageView:

<ImageView

android:id="@+id/icon\_iv"

android:layout\_width="48dp"

android:layout\_height="48dp"

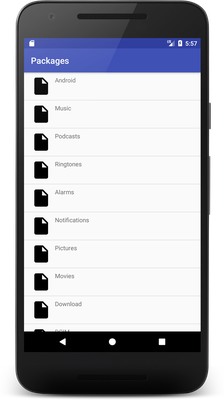
android:adjustViewBounds="true"

android:src="@drawable/ic\_file\_92dp" />

Я немного промахнулся с размерами, указав 92dp — 48 в самый раз. Благо, мы можем без проблем задать размеры прямо в ImageView.

Проделайте то же самое с view\_item\_directory.xml, но используйте иконку для директории.

После этого запустите приложение:

[](https://skillberg.com/media/uploads/2017/11/17/81f834f4c009456f9a8005d1613b2cf9.png)Список файлов в RecyclerView

Хм, непорядок. Все, что видно на скриншоте — директории, а показываются файлы. Почему? Потому что мы ещё не реализовали обработку различающихся типов данных, чем сейчас и займёмся.

ItemViewType в RecyclerView

Как я сказал ранее, мы можем различать типы данных в адаптере RecyclerView. Благодаря этому мы можем создавать различные View для каждого типа данных, которые у нас есть. Откройте адаптер.

Для начала для удобства создадим две константы, которые будут отвечать за два типа данных: файл и директорию

private static final int TYPE\_DIRECTORY = 0;

private static final int TYPE\_FILE = 1;

Теперь переопределим метод getItemViewType(). В него в качестве параметра передается индекс элемента, по этому индексу мы получаем элемент и возвращаем соответствующую ему константу типа данных:

@Override

public int getItemViewType(int position) {

File file = files.get(position);

if (file.isDirectory()) {

return TYPE\_DIRECTORY;

} else {

return TYPE\_FILE;

}

}

Теперь нужно немного дополнить метод onCreateViewHolder(), в котором создаётся View для соответствующей ячейки. В него вторым параметром передаётся тип данных, который адаптер берёт как раз из метода getItemViewType(). В соответствии с типом данных мы будем создавать View для файла или директории:

@Override

public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {

LayoutInflater layoutInflater = LayoutInflater.from(parent.getContext());

View view;

if (viewType == TYPE\_DIRECTORY) {

view = layoutInflater.inflate(R.layout.view\_item\_directory, parent, false);

} else {

view = layoutInflater.inflate(R.layout.view\_item\_file, parent, false);

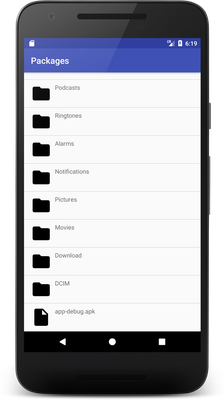
}

return new ViewHolder(view);

}

Наш случай весьма прост, и поэтому данного кода будет достаточно. В более сложных приложениях может потребоваться изменить onBindViewHolder() и, возможно, добавить ещё несколько ViewHolder-ов в случае, если лэйауты сильно отличаются. В нашем же случае лэйауты отличаются лишь картинкой, поэтому делать этого не придётся.

Запустите приложение, и увидите, что теперь директории и файлы отображаются корректно:

[](https://skillberg.com/media/uploads/2017/11/17/a18c756f36ab408396ac806bd0f7a164.png)Отображение файлов и директорий

Обработка кликов в RecyclerView

Теперь мы должны добавить обработку кликов на элементы списка. В случае клика на директорию приложение должно открыть соответствующую директорию, а если пользователь кликнет на файл — выбрать файл, если тип файла соответствует нужному нам.

Каждый View в Android может иметь тэг. Не стоит путать его с XML-тэгами: тэг у View — это объект, ассоциированный с ним. С помощью тэга мы можем "сохранить" во View любые данные.

Конкретно в нашем случае он нужен, чтобы ассоциировать ячейку списка с конкретным файлом, который на данный момент отображается в ячейке. Зададим тэг в методе onBindViewHolder():

@Override

public void onBindViewHolder(ViewHolder holder, int position) {

File file = files.get(position);

holder.nameTv.setText(file.getName());

holder.itemView.setTag(file);

}

Теперь дополним конструктор ViewHolder, добавив туда OnClickListener:

public ViewHolder(View itemView) {

super(itemView);

nameTv = itemView.findViewById(R.id.name\_tv);

itemView.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View view) {

}

});

}

В методе onClick() мы получаем View, на который кликнул пользователь. В нём же мы можем получить тэг этого View, а соответственно и файл, который был установлен в качестве тэга:

File file = (File) view.getTag();

Обратите внимание, что тэг имеет тип Object, поэтому нам нужно привести его к типу File.

Ок, теперь мы знаем, как получить файл, на который произошёл клик, однако, с ним нужно ещё и что-то сделать. Хорошей практикой считается обработка кликов в Activity, получается, нам нужно передать событие клика в Activity. Чтобы сделать это, добавьте в самый конец класса адаптера вот такой интерфейс:

public interface OnFileClickListener {

void onFileClick(File file);

}

В сам адаптер добавьте поле и setter для него:

@Nullable

private OnFileClickListener onFileClickListener;

public void setOnFileClickListener(@Nullable OnFileClickListener onFileClickListener) {

this.onFileClickListener = onFileClickListener;

}

Аннотация @Nullable говорит о том, что поле вполне вероятно может быть null, и чтобы избежать неожиданностей лучше проверять его на null перед использованием.

Теперь в метод onClick() в конструкторе ViewHolder добавьте следующие строчки:

if (onFileClickListener != null) {

onFileClickListener.onFileClick(file);

}

Готово, теперь при клике на файл будет "выстреливаться" событие onFileClick() нашего onFileClickListener, если он, конечно, не будет null. Осталось добавить обработку этого события в Activity.

Добавьте следующий код в конец класса FilePickerActivity:

private final FilesAdapter.OnFileClickListener onFileClickListener = new FilesAdapter.OnFileClickListener() {

@Override

public void onFileClick(File file) {

}

};

Теперь нужно установить этот Listener в адаптер. Помните мы рассматривали жизненный цикл Activity? Listener лучше всего ставить в методе onStart(), а убирать — в методе onStop(). Зачем убирать, спросите вы? Чтобы система не отправила запоздалый клик в Activity в тот момент, когда она уже не будет отображаться на экране. Вообще, очень важно отписываться от всех событий в момент, когда Activity заканчивает свою работу.

@Override

protected void onStart() {

super.onStart();

filesAdapter.setOnFileClickListener(onFileClickListener);

}

@Override

protected void onStop() {

filesAdapter.setOnFileClickListener(null);

super.onStop();

}

Обратите внимание на порядок вызова методов суперкласса в onStart() и onStop() — если подписываться мы должны после выполнения метода суперкласса, то отписываться — до.

Теперь в методе onFileClick() нашего Listener-а добавьте следующий код:

if (file.isDirectory()) {

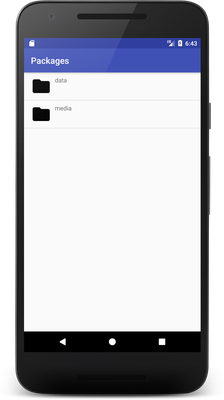
fileManager.navigateTo(file);

updateFileList();

}

В нём мы проверяем, нажал ли пользователь на директорию, и, если да — переходим в неё и обновляем список файлов в RecyclerView.

Если вы запустите приложение сейчас, и попробуете перейти, скажем, в директорию Android, то увидите следующее:

[](https://skillberg.com/media/uploads/2017/11/17/4092f2ad954042f0b0de994fe19a7638.png)Переход в директорию Android

Это значит, что все работает!

Обработка onBackPressed()

Однако, есть одна проблемка: перейти-то в директорию мы можем, а вот вернуться назад — нет. Ранее мы условились, что будем переходить в директорию выше нажатием на кнопку **Назад**.

Соответственно, нужно как-то перехватить событие нажатия на эту кнопку. Это во-первых. А во-вторых, вы могли заметить, что система при нажатии на эту кнопку возвращает нас к предыдущей Activity.

Значит, нужно сделать что-то такое, чтобы она возвращала нас к предыдущей директории только в том случае, если выше подняться мы уже не можем.

К счастью, сделать это очень просто. У Activity есть метод onBackPressed(), который вызывается каждый раз, когда пользователь нажимает на кнопку **Назад**. Переопределим его:

@Override

public void onBackPressed() {

super.onBackPressed();

}

Теперь вставим в него следующий код:

if (fileManager != null && fileManager.navigateUp()) {

updateFileList();

} else {

super.onBackPressed();

}

Таким образом мы вызываем метод суперкласса только в том случае, если fileManager ещё не существует (например, если пользователь не предоставил разрешение), или если не можем подняться на директорию выше).

Именно в этих случаях произойдёт возврат к предыдущей Activity. Если же мы поднялись на директорию выше, то обновляем список файлов в RecyclerView.

Попробуйте запустить приложение и погулять по директориям.

Обработка типа файла

Осталось лишь среагировать на клик на файл, и понять, APK ли нам дали. В принципе, для этих целей вполне сойдёт проверить расширение, система в любом случае будет проверять файл перед установкой.

private final FilesAdapter.OnFileClickListener onFileClickListener = new FilesAdapter.OnFileClickListener() {

@Override

public void onFileClick(File file) {

if (file.isDirectory()) {

fileManager.navigateTo(file);

updateFileList();

} else {

if (file.getName().endsWith(".apk")) {

// Делаем что-то с файлом

}

}

}

};

Ок, готово. Но как теперь передать файл в Activity?

Передача данных в вызывающую Activity. startActivityForResult()

В Android существует механизм, позволяющий не только запустить Activity, но и получить из неё какой-то результат. Нам нужно передать одну строку — путь до выбранного файла.

Для того, чтобы добиться этого, потребуется немного дополнить FilePickerActivity и MainActivity.

Начнём с FilePickerActivity. В ней при выборе корректного файла мы должны создать специальный Intent, в него положить данные, которые требуется передать в MainActivity и вызвать метод setResult().

Для передачи данных через Intent существует такой класс, как Bundle. Это достаточно простое временное хранилище пар "ключ": "значение" — мы кладём туда данные, используя нужные нам ключи, и потом получаем по этим же ключам.

Ключи принято выносить в отдельные строковые константы. Добавьте в самый верх класса FilePickerActivityследующую строчку:

public static final String EXTRA\_FILE\_PATH = "file\_path";

Поле объявлено публичным, чтобы мы смогли получить к нему доступ из MainActivity.

Теперь дополните метод onFileClick():

@Override

public void onFileClick(File file) {

if (file.isDirectory()) {

fileManager.navigateTo(file);

updateFileList();

} else {

if (file.getName().endsWith(".apk")) {

Intent intent = new Intent();

intent.putExtra(EXTRA\_FILE\_PATH, file.getAbsolutePath());

setResult(RESULT\_OK, intent);

finish();

}

}

}

В случае, если выбран корректный файл, мы создаём Intent, засовываем в него путь до файла (Extra — это "обёртка" для Bundle, того самого хранилища. Нужна для более удобного доступа).

Далее вызываем setResult(), в котором сообщаем, что все прошло хорошо, и передаём созданный Intent.

Кстати, есть перегруженный вариант метода setResult(), в котором единственным аргументом является код — для случаев, когда никаких данных передавать не надо, а нужно лишь сообщить результат выполнения задачи.

После этого мы завершаем Activity вызовом метода finish().

Теперь нужно "поймать" результат в MainActivity.

Для этого, сначала нужно немного изменить запуск FilePickerActivity — заменить метод на startActivityForResult(). Этим методом мы говорим системе, что не просто запускаем Activity, но и ждём какого-то результата от неё.

Помимо Intent в этот метод так же передаётся параметр requestCode. Код запроса нужен для тех случаев, когда мы запускаем таким способом несколько Activity, и потом хотим понять, от какой именно Activityпришёл результат.

Объявим в самом начале MainActivity следующую константу:

private static final int REQUEST\_CODE\_PICK\_APK = 1;

Теперь заменим вызов startActivity() на startActivityForResult():

private void startFilePickerActivity() {

Intent intent = new Intent(this, FilePickerActivity.class);

startActivityForResult(intent, REQUEST\_CODE\_PICK\_APK);

}

Последний шаг — переопределить метод onActivityResult(). В нём мы проверим, что ответ пришёл именно от нашей Activity, получим путь до файла и выведем его в лог:

@Override

protected void onActivityResult(int requestCode, int resultCode, Intent data) {

if (requestCode == REQUEST\_CODE\_PICK\_APK && resultCode == RESULT\_OK) {

String apkPath = data.getStringExtra(FilePickerActivity.EXTRA\_FILE\_PATH);

Log.i(TAG, "APK: " + apkPath);

} else {

super.onActivityResult(requestCode, resultCode, data);

}

}

Запускаем, выбираем APK-файл, видим в логах:

09-10 19:17:03.446 com.test.packages I/MainActivity: APK: /storage/emulated/0/app-debug.apk

Работает!

Установка APK

Самое последнее, что осталось сделать — это, собственно, установить APK. Самостоятельно мы не можем сделать этого из-за ограничений безопасности Android, но мы можем попросить систему сделать это за нас. В этом нам как раз помогут неявные Intent.

Всё, что требуется сделать — запустить Activity, с помощью которой можно открыть APK файл. Почти то же самое, что мы делали ранее, но немного отличающееся. Дело в том, что мы не знаем, какая именно Activityотвечает за открытие файлов APK.

Тут-то на помощь и приходят неявные Intent.

Отличие от явных, которые мы использовали ранее, в том, что мы говорим, что именно и с чем мы хотим сделать, а система (возможно, при помощи пользователя) выбирает, какое приложение и какую Activity для этого использовать.

До Android 7.0 все было достаточно просто:

Intent installIntent = new Intent(Intent.ACTION\_VIEW);

installIntent.setDataAndType(Uri.fromFile(new File(apkPath)), "application/vnd.android.package-archive");

installIntent.setFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK); // Создаст новый процесс

startActivity(installIntent);

Выполнив этот код, мы говорим системе, что хотим установить APK, и она показывает пользователю диалог установки. Однако, если вы выполните этот код на Android 7.0 или выше, то приложение просто упадёт примерно с таким стэк трейсом:

Caused by: android.os.FileUriExposedException: file:///storage/emulated/0/app-debug.apk exposed beyond app through Intent.getData()

at android.os.StrictMode.onFileUriExposed(StrictMode.java:1799)

at android.net.Uri.checkFileUriExposed(Uri.java:2346)

at android.content.Intent.prepareToLeaveProcess(Intent.java:8965)

at android.content.Intent.prepareToLeaveProcess(Intent.java:8926)

at android.app.Instrumentation.execStartActivity(Instrumentation.java:1517)

at android.app.Activity.startActivityForResult(Activity.java:4225)

at android.support.v4.app.BaseFragmentActivityApi16.startActivityForResult(BaseFragmentActivityApi16.java:54)

at android.support.v4.app.FragmentActivity.startActivityForResult(FragmentActivity.java:65)

at android.app.Activity.startActivityForResult(Activity.java:4183)

at android.support.v4.app.FragmentActivity.startActivityForResult(FragmentActivity.java:711)

at android.app.Activity.startActivity(Activity.java:4522)

at android.app.Activity.startActivity(Activity.java:4490)

at com.test.packages.MainActivity.onActivityResult(MainActivity.java:105)

at android.app.Activity.dispatchActivityResult(Activity.java:6932)

at android.app.ActivityThread.deliverResults(ActivityThread.java:4085)

at android.app.ActivityThread.handleSendResult(ActivityThread.java:4132)

at android.app.ActivityThread.-wrap20(ActivityThread.java)

at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1533)

at android.os.Handler.dispatchMessage(Handler.java:102)

at android.os.Looper.loop(Looper.java:154)

at android.app.ActivityThread.main(ActivityThread.java:6119)

at java.lang.reflect.Method.invoke(Native Method)

at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:886)

Дело в том, что в Android 7.0 изменился подход к передаче данных. Мы передаем ссылку на файл в виде локального URI, а система ждёт от нас ссылки, переданной через FileProvider.

FileProvider

В Android существует механизм ContentProvider. Как вы помните из самого первого урока, данные каждого приложения хранятся в его собственной директории, доступ к которой может иметь только само приложение.

Представьте приложение "Контакты". Оно хранит все контакты в своей базе данных, доступ к которой есть только у него. А приложениям "Телефон" и "Смс" обязательно нужно получить доступ к контактам.

ContentProvider нужны как раз для того, чтобы предоставить этот доступ, не нарушая при этом паттернов обеспечения безопасности.

Сейчас я не буду подробно останавливаться на ContentProvider-ах, мы рассмотрим их позже. Просто запомните, что это такая штука, которая предоставляет доступ к своим данным для сторонних приложений.

Собственно, в определённый момент в Google решили, что отправка обычных локальных URI, содержащих адрес файла — это неправильно, и заставили разработчиков использовать для этих целей FileProvider, который является сабклассом ContentProvider.

Ну что ж, давайте разбираться, как это починить.

В первую очередь нужно задать список путей, которые мы разрешим использовать в FileProvider. Для этого создайте в ресурсах директорию xml, а в ней файл provider\_paths.xml. В файл поместите следующий контент:

<?xml version="1.0" encoding="utf-8"?>

<paths>

<external-path

name="external\_files"

path="." />

</paths>

В нём мы говорим, что предоставляем доступ к SD-карте.

Далее, добавьте провайдер в AndroidManifest.xml, сразу после последней Activity (перед закрывающим тегом </application>:

<provider

android:name="android.support.v4.content.FileProvider"

android:authorities="${applicationId}.provider"

android:exported="false"

android:grantUriPermissions="true">

<meta-data

android:name="android.support.FILE\_PROVIDER\_PATHS"

android:resource="@xml/provider\_paths" />

</provider>

Gradle автоматически подставит имя пакета вместо ${applicationId}, получится что-то вроде com.test.packages.provider.

Теперь в коде MainActivity создайте метод startAppInstallation():

private void startAppInstallation(String apkPath) {

}

Именно в нём мы будем запускать установку приложения.

В сам метод добавьте следующие строки:

Intent installIntent = new Intent(Intent.ACTION\_VIEW);

Uri uri;

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.N) {

uri = FileProvider.getUriForFile(this,

BuildConfig.APPLICATION\_ID + ".provider", new File(apkPath));

installIntent.addFlags(Intent.FLAG\_GRANT\_READ\_URI\_PERMISSION);

} else {

uri = Uri.fromFile(new File(apkPath));

}

installIntent.setDataAndType(uri, "application/vnd.android.package-archive");

installIntent.addFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK); // Создаст новый процесс

startActivity(installIntent);

Что за странные if-ы, спросите вы? Как вы помните, механизм FileProvider появился начиная с Android 7 (N), соответственно, мы не можем использовать его на более ранних версиях. Но на более ранних версиях мы вполне можем использовать старый способ создания URI.

Поэтому мы проверяем версию Android, и если она больше либо равна N (7.0), то получаем Uri через FileProvider, а также добавляем флаг, дающий право на чтение для внешнего приложения к Intent, а в ином случае — просто создаём Uri по-старинке.

Далее мы устанавливаем Uri в Intent и передаем тип данных. application/vnd.android.package-archive— это тип данных, соответствующий APK-файлу.

После этого запускаем Activity.

Последний шаг — изменяем onActivityResult():

@Override

protected void onActivityResult(int requestCode, int resultCode, Intent data) {

if (requestCode == REQUEST\_CODE\_PICK\_APK && resultCode == RESULT\_OK) {

String apkPath = data.getStringExtra(FilePickerActivity.EXTRA\_FILE\_PATH);

Log.i(TAG, "APK: " + apkPath);

startAppInstallation(apkPath);

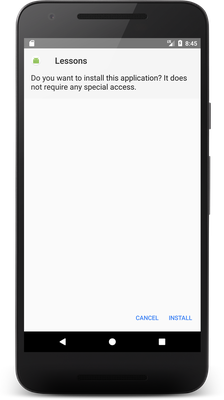
} else {

super.onActivityResult(requestCode, resultCode, data);

}

}

Запустите приложение, выберите файл, и увидите появившийся системный диалог, в котором нас спросят, хотим ли мы установить приложение.

[](https://skillberg.com/media/uploads/2017/11/17/948d469324294be9a2abb4614b9d1b66.png)Установка приложения при помощи Intent

Мы восхитительны!

ПРИМЕРЫ КОДА

[ИСХОДНЫЙ КОД УРОКА](https://github.com/Skillberg/PackagesAndroid/tree/lesson-17)