**Файловая система для приложения**

[Метод getFilesDir() - где мы](http://developer.alexanderklimov.ru/android/filesystem.php#getfilesDir)  
[Файловая система Android](http://developer.alexanderklimov.ru/android/filesystem.php#common)  
[Метод openFileOutput() - Сохраняем файл](http://developer.alexanderklimov.ru/android/filesystem.php#savefile)  
[Метод fileList() - Получаем список сохранённых файлов](http://developer.alexanderklimov.ru/android/filesystem.php#filelist)  
[Метод openFileInput() - Открываем файл для чтения](http://developer.alexanderklimov.ru/android/filesystem.php#readfile)  
[Метод deleteFile() - Удаляем файл](http://developer.alexanderklimov.ru/android/filesystem.php#deletefile)  
[Метод getCacheDir() - Кэш для временных файлов](http://developer.alexanderklimov.ru/android/filesystem.php#cache)

**Файловая система Android для приложения**

Android использует файловую систему, основанную на Linux. Файлы хранятся в энергонезависимой флеш-памяти, встроенной в устройство, поэтому они не теряются, когда телефон выключают. Также многие устройства поддерживают встроенную карточку памяти.

Есть два варианта работы с файлами. Первый - это стандартные методы, принятые в Java, которые можно использовать, например, для работы с файлами и папками на внешнем накопителе. Это [стандартные функции ввода/вывода](http://developer.alexanderklimov.ru/android/io.php) из Java (пакет **java.io**).

Также существуют специальные Android-версии методов с файлами, которые относятся к отдельным приложениям. Необходимо помнить, что доступ к файлам приложений ограничен, чтобы вы не могли вносить беспорядок в данные других приложений.

Вкратце рассмотрим несколько стандартных методов. У класса **android.os.Environment** есть методы для работы с папками и файлами. Все методы возвращают тип **File**, который может быть как файлом, так и папкой. Чтобы понять, с чем мы имеем дело, нужно вызвать метод **isFile()** или **isDirectory()** класса **File**.

Класс **Environment** имеет несколько удобных методов для чтения стандартных папок Android:

* getDataDirectory() - папка для хранения данных
* getDownloadCachedDirectory() - папка для хранения загружаемых внешних файлов и кэша
* getRootDirectory() - корневая папка файловой системы Android

На практике, вам практически не придётся использовать эти методы. Также есть методы, связанные с внешним накопителем, о которых поговорим в других статьях.

В Android дополнительно реализована своя система работы с файлами. Основное место для доступа у вашей программы - это приватная папка пакета, созданная во время установки (data/data/packagename). Например, файлы сохраняются в подпапке **files**.

Строго говоря, при работе с файлами желательно использовать многопоточность. Но пока явно среда разработки не запрещает работать с файлами в основном потоке, поэтому примеры мы не будем усложнять.

В классе **Context** есть несколько методов, которые позволяют читать и записывать данные в эту папку:

* **fileList()** - возвращает список файлов в приватной зоне приложения в массиве String
* **getFilesDir()** - возвращает абсолютный путь к файлу приложения
* **getFileStreamPath()**
* **getDir()** - возвращает объект **File**, который содержит путь к директории приложения
* **getExternalFilesDir** - возвращает объект **File**, который содержит путь к директории приложения на внешнем накопителе
* **openFileInput()** - открывает приватный файл для чтения, возвращает java.io.FileInputStream
* **openFileOutput()** - открывает приватный файл для записи. Возвращает java.io.FileOutputStream
* **deleteFile()** - удаляет приватный файл. Возвращает true при успешном завершении действия, в противном случае возвращается false
* **getCacheDir()** - путь к кэшу для хранения временных файлов
* **getExternalCacheDir()** - путь к кэшу для хранения временных файлов на внешнем носителе

Часть этих методов мы рассмотрим ниже. Про остальные читайте в документации.

**Совместный доступ**

Каждое приложение при установке получает собственный идентификатор (ID), который позволяет читать и записывать данные в приватной папке приложения. Однако, если два приложения подписаны одним и тем же цифровым сертификатом, Android считает, что они поступили от одного и того же разработчика, и даёт им один и тот же идентификатор. Это позволяет разделять различные виды данных друг с другом, если это необходимо. С другой стороны, вы должны позаботиться, что совместный доступ не помешал им.

**Метод getFilesDir() - где мы**

Напомню, что файлы приложения хранятся в директории **data/data/имя\_пакета/files**. Вы можете в этом убедиться с помощью метода **getFilesDir()**, который возвращает абсолютный путь к файлу.

File file = getFilesDir();

Log.i("File", file.toString());

// возвратит /data/data/ru.alexanderklimov.testapplication/files

А вот с методом **getDir()** немного запутанней. Запустим переделанный код из предыдущего примера:

File file = getDir("cat", MODE\_PRIVATE);

Log.i("File", file.toString());

В результате мы увидим строку **data/data/имя\_пакета/app\_cat**. Я не стал бы рекомендовать использовать данный метод, да и на практике его не встречал.

**Метод openFileOutput() - Сохраняем** **файл**

Если при использовании метода **FileOutputStream()** файл, который вы указали, не существует, Android создаст его для вас. По умолчанию имеющиеся файлы перезаписываются. Чтобы добавить данные в конец существующего файла, установите режим **Context.MODE\_APPEND**.

По умолчанию файлы, созданные с помощью метода **openFileOutput()**, принадлежат только текущему приложению, другие приложения не будут иметь к нему доступ. Стандартный способ открыть доступ к файлу разным приложениям — использование класса **ContentProvider**. Еще один вариант — указать при создании файла режим **Context.MODE\_WORLD\_READABLE**или **Context.MODE\_WORLD\_WRITEABLE** (считается устаревшим с API 17), как показано в следующем фрагменте кода:

String OUTPUT\_FILE = "public\_cat.txt";

FileOutputStream fos = openFileOutput(OUTPUT\_FILE, Context.MODE\_WORLD\_WRITEABLE);

Предположим, вы хотите написать заметку о том, сколько раз мяукал ваш кот воскресным утром, не давая выспаться, и сохранить её себе на память. Вам необходимо создать новый приватный файл, записать туда данные и закрыть его. Создадим два текстовых поля - в одном будем записывать имя файла, а в другом будем писать данные. А также добавим кнопку для сохранения записанной заметки:

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

Button saveFileButton = (Button) findViewById(R.id.buttonSave);

final EditText fileNameEditText = (EditText) findViewById(R.id.editTextFileName);

final EditText contentEditText = (EditText) findViewById(R.id.editTextContent);

saveFileButton.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

String fileName = fileNameEditText.getText().toString(); // имя будущего файла

String content = contentEditText.getText().toString(); // получаем содержимое

FileOutputStream fos;

try {

fos = openFileOutput(fileName, Context.MODE\_PRIVATE); // открываем файл для записи

fos.write(content.getBytes()); // записываем данные

fos.close(); // закрываем файл

// выводим сообщение

Toast.makeText(getApplicationContext(),

"Файл " + fileName + " сохранён", Toast.LENGTH\_LONG).show();

} catch (FileNotFoundException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (IOException e) {

// TODO Auto-generated catch block

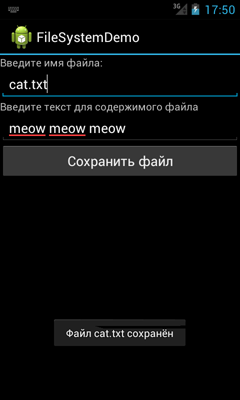
e.printStackTrace();

}

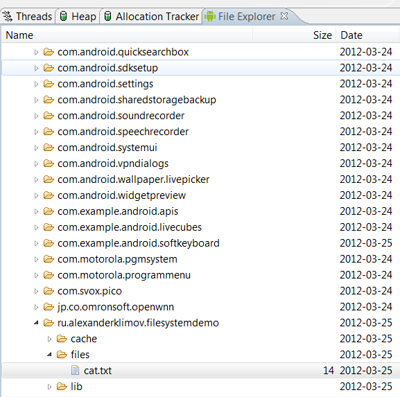
}

});

}



Вы можете запустить **Android Device Monitor** и в панели **File Explorer** найти созданный файл. В моём случае файл находится по пути**/data/data/ru.alexanderklimov.filesystemdemo/files/cat.txt**.



На самом деле создавать файл с расширением необязательно, как мы привыкли это делать в Проводнике Windows, так как файлы скрыты от всех и пользователь не будет шастать по папкам приложений. Я использовал расширение TXT по привычке.

Также не забывайте закрывать файловый поток!

**Метод fileList() - Получаем список сохранённых файлов**

Но, нам совсем не обязательно использовать вкладку **File Explorer** для получения списка файлов. Мы и сами с усами - и можем написать код, отображающий список файлов. Добавим к предыдущему примеру еще одну кнопку и список. Теперь напишем следующее:

Button getFileListButton = (Button) findViewById(R.id.buttonGetFileList);

final ListView savedFilesListView = (ListView) findViewById(R.id.listViewFileList);

getFileListButton.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

String[] savedFilesArray = getApplicationContext().fileList(); // получаем массив имен файлов

// передаем массив адаптеру для вывода в списке

ArrayAdapter <String> adapter = new ArrayAdapter<>(

v.getContext(), //getApplicationContext(),

android.R.layout.simple\_list\_item\_1, savedFilesArray);

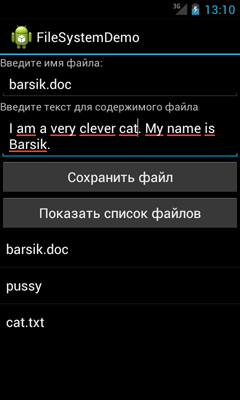
// выводим список файлов

savedFilesListView.setAdapter(adapter);

}

});

Как видите, для получения списка сохраненных файлов достаточно одной строчки кода с вызовом метода **fileList**.



**Метод openFileInput() - Открываем** **файл для чтения**

Мы можем сохранять файлы и получить список сохраненных файлов. Теперь мы должны открыть любой из этих файлов и посмотреть, что там написано. Возможно там хранится ответ на главный вопрос человечества: Кто сказал мяу?

На этот раз мы не будем добавлять в проект новую кнопку. У нас есть элемент **ListView**, содержащий список файлов. Добавим обработку щелчка на пункте списка для открытия конкретного файла.

savedFilesListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {

@Override

public void onItemClick(AdapterView<?> parent, View view,

int position, long id) {

String clickedFile = (String) parent

.getItemAtPosition(position);

openFileDialog(clickedFile);

}

});

private void openFileDialog(String file) {

// чтение файла из файлового хранилища

FileInputStream fis;

String content = "";

try {

fis = openFileInput(file); // открываем файл для чтения

byte[] input = new byte[fis.available()];

while (fis.read(input) != -1) {

content += new String(input);

}

fis.close();

} catch (FileNotFoundException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

// создаем собственное диалоговое окно

AlertDialog.Builder fileDialog = new AlertDialog.Builder(

this);

fileDialog.setTitle(file);

TextView contentTextView = new TextView(this);

contentTextView.setText(content);

ViewGroup.LayoutParams textViewLayoutParams = new ViewGroup.LayoutParams(

ViewGroup.LayoutParams.WRAP\_CONTENT, ViewGroup.LayoutParams.WRAP\_CONTENT);

contentTextView.setLayoutParams(textViewLayoutParams);

fileDialog.setView(contentTextView);

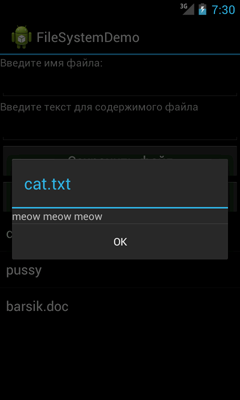
fileDialog.setPositiveButton("OK", null);

fileDialog.show();

}

В этом примере мы вызываем метод **openFileInput()**, чтобы открыть файл для чтения и выводим результат в диалоговом окне.

Принцип простой - после открытия файла мы начинаем построчно считывать данные в цикле **while**. Когда строки закончатся, то метод **read()** вернёт значение -1 и можно выходить из цикла. Все считанные данные накапливаются в строковой переменной **content**, которую выводим в **TextView**. Не забываем закрывать файловый поток.



**Метод deleteFile() - Удаляем файл**

Осталась еще одна распространённая операция с файлами - удаление. Удаляется файл очень просто - раз и готово. Чтобы не перезагружать интерфейс нашей программы лишними деталями, повесим процедуру удаления файла на событие длительного нажатия в списке. В реальном приложении так делать не следует, в противном случае пользователи на вас сильно обидятся. Стоит им на секунду задержать лапу на элементе списка - и файла как не бывало.

После удаления файла неплохо бы также обновить список сохранённых файлов. У нас уже есть код для кнопки. Можно вынести данный код в отдельный метод **showSavedFiles()** и использовать его также и при удалении файла для обновления списка.

private void showSavedFiles() {

ListView savedFilesListView = (ListView) findViewById(R.id.listViewFileList);

String[] savedFilesArray = getApplicationContext().fileList();

ArrayAdapter adapter = new ArrayAdapter<>(this,

android.R.layout.simple\_list\_item\_1, savedFilesArray);

savedFilesListView.setAdapter(adapter);

}

savedFilesListView.setOnItemLongClickListener(new AdapterView.OnItemLongClickListener() {

@Override

public boolean onItemLongClick(AdapterView<?> parent,

View view, int position, long id) {

String longClickedFile = (String) parent

.getItemAtPosition(position);

// удаляем выбранный файл

deleteFile(longClickedFile);

Toast.makeText(getApplicationContext(),

longClickedFile + " удалён", Toast.LENGTH\_LONG)

.show();

showSavedFiles();

return true;

}

});

**Метод getCacheDir() - Кэш для** **временных файлов**

Если вашему приложению нужно кэшировать временные файлы, то можно использовать внутренний управляемый и внешний неуправляемый кэш. Файлы, хранящие в кэше, будут стёрты системой при нехватке места на внутреннем носителе. Так как система не отслеживает количества места на внешнем носителе, то там кэшированные файлы удаляться не будут (во всяком случае не гарантируется), поэтому следите за мусором самостоятельно.

File file = getCacheDir();

Toast.makeText(this, file.toString(), Toast.LENGTH\_LONG).show();

Возвращает строку **data/data/имя\_пакета/cache**.

Для внешнего кэша используется метод **getExternalCacheDir()**.