

Handheld App Development

Lec 6: Saving data I (File)

Ekarat Rattagan, PhD

Outline

- Saving Key-Value Sets
- **Saving Files**
- Saving Data in SQL Databases

Saving Files

- Android uses disk-based file systems.
- A File object is suited to read or write large amounts of data, e.g., image files or anything exchanged over a network.
- API: Java.io

Internal or External Storage

Two file storage areas

- **Internal storage:** built-in non-volatile memory
- **External storage:** a removable storage medium (micro SD card)

Some devices divide the permanent storage space into two storage spaces (even without a removable storage medium)

Internal or External Storage

Internal storage	External storage
Always available.	not always available
Files saved here are accessible by only you app.	files saved here may be read outside of your control
When the user uninstalls your app, the system removes all your app's files from internal storage.	When the user uninstalls your app, the system removes your app's files from here only if you save them in the directory from <code>getExternalFilesDir()</code> .
best when you want to be sure that neither the user nor other apps can access your files.	best place for files that don't require access restrictions and for files that you want to share with other apps or allow the user to access with a computer.

Permissions

- Permissions for External Storage

```
<manifest ...>  
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"  
/>  
    <uses-permission  
android:name="android.permission.READ_EXTERNAL_STORAGE" />  
</manifest>
```

Don't need any permissions to save files on the internal storage. Your application always has permission to read and write files in its internal storage directory.

Save a File on Internal Storage

```
String filename    = "myfile";
String string      = "Hello world!";
File file          = new File(context.getFilesDir(), filename);

try
{
    FileOutputStream outputStream = openFileOutput(filename,
Context.MODE_PRIVATE);
    outputStream.write(string.getBytes());
    outputStream.close();
}
catch (Exception e)
{
    e.printStackTrace();
}
```

Read a File on Internal Storage

```
try {
    InputStream inputStream = mContext.openFileInput(fileName);
    if ( inputStream != null ) {
        InputStreamReader inputStreamReader = new InputStreamReader(inputStream);
        BufferedReader bufferedReader = new BufferedReader(inputStreamReader);
        String receiveString = "";
        StringBuilder stringBuilder = new StringBuilder();

        while ( (receiveString = bufferedReader.readLine()) != null )
        {
            stringBuilder.append(receiveString); }

        inputStream.close();
        ret = stringBuilder.toString();
    }
} catch (IOException e) {
    Log.e("login activity", "Can not read file: " + e.toString());
}
```


Example 1

- See the attached file (Ex1.pdf)
- In Ex1.pdf, you will see the new View (SnackBar)
- **To use SnackBar**
 - Insert `compile 'com.android.support.design:25.1.1'`
“
into `build.gradle > dependencies` and `sync now`
- **More about SnackBar**
 - <http://www.akexorcist.com/2015/07/android-design-support-library-snackbar.html>

Write & Read a File on External Storage

```
File file = new File(mContext.getExternalFilesDir(null), fileName);
```

```
/* Checks if external storage is available for read and write */
public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        return true;
    }
    return false;
}
```

Because the external storage may be unavailable—such as when the user has mounted the storage to a PC or has removed the SD card that provides the external storage—you should always verify that the volume is available before accessing it. You can query the external storage state by calling [getExternalStorageState\(\)](#). If the returned state is equal to [MEDIA_MOUNTED](#), then you can read and write your files. For example, the following methods are useful to determine the storage availability:

More about External Storage

- To check the size of external storage

```
long freeSize = mContext.getExternalFilesDir(null).getFreeSpace();  
long totalSize = mContext.getExternalFilesDir(null).getTotalSpace();
```

However, the system does not guarantee that you can write as many bytes as are indicated by [`getFreeSpace\(\)`](#).

- If the number returned is a few MB more than the size of the data you want to save, or
- if the file system is less than 90% full, then it's probably safe to proceed.

Otherwise, you probably shouldn't write to storage.

Exercise 1

- 1.1 Modify **the writeFile method**,
- to get the percentage of free space size?
 - to allow to save file, if after saving, the percentage of free space size is more than 10%.

Delete a File

You should always delete files that you no longer need. The most straightforward way to delete a file is to have the opened file reference call [`delete\(\)`](#) on itself.

```
myFile.delete();
```

If the file is saved on internal storage, you can also ask the [`Context`](#) to locate and delete a file by calling [`deleteFile\(\)`](#):

```
myContext.deleteFile(fileName);
```

Cache a file

```
File file = File.createTempFile(filename, null,  
this.getCacheDir());
```

- To add a random number on the end of the filename to keep files unique
- Cache file, e.g., file downloaded from internet such as image, etc.

```
public File getTempFile(Context context, String url) {  
    File file;  
    try {  
        String fileName = Uri.parse(url).getLastPathSegment();  
        file = File.createTempFile(fileName, null, context.getCacheDir());  
    } catch (IOException e) {  
        // Error while creating file  
    }  
    return file;  
}
```

Exercise 2

2.1 Implement saving a cache file & observe the cache file generated

Save public files on the external storage

If you want to save public files on the external storage, use the [getExternalStoragePublicDirectory\(\)](#) method to get a [File](#) representing the appropriate directory on the external storage. The method takes an argument specifying the type of file you want to save so that they can be logically organized with other public files, such as [DIRECTORY MUSIC](#) or [DIRECTORY PICTURES](#). For example:

```
public File getAlbumStorageDir(String albumName) {  
    // Get the directory for the user's public pictures directory.  
    File file = new  
File(Environment.getExternalStoragePublicDirectory(  
        Environment.DIRECTORY_PICTURES), albumName);  
    if (!file.mkdirs()) {  
        Log.e(LOG_TAG, "Directory not created");  
    }  
    return file;  
}
```

* If you want to save files that are private to your app, you can acquire the appropriate directory by calling [getExternalFilesDir\(\)](#)

More file types

- Load an image from SD Card

```
File    sdCard = Environment.getExternalStorageDirectory();
File    directory = new File (sdCard.getAbsolutePath() + "/"
Your_Path");
File    file = new File(directory, "image_name.jpg");
Bitmap  bitmap = BitmapFactory.decodeFile(file);

//Set to View
ImageView myImageView = (ImageView)findViewById(R.id.imageview);

myImageView.setImageBitmap(bitmap);
```

Android External Library

- <https://github.com/eluleci/FlatUI>



<https://codecanyon.net/item/android-clean-flat-ui-components/11316073>

Interesting External Library for files

- Encrypt/Decrypt library
 - <https://github.com/tozny/java-aes-crypto>

Constraint Layout

<https://codelabs.developers.google.com/codelabs/constraint-layout/index.html?index=..%2F..%2Findex#0>

Keyword to search: **codelabs constraint layout**