

2 SAVING FILES ON EXTERNAL STORAGE

Sources:

<http://developer.android.com/training/basics/data-storage/files.html#WriteExternalStorage>

<http://developer.android.com/reference/java/io/File.html>

Android uses a file system that's similar to disk-based file systems on other platforms. This lesson describes how to work with the Android file system to read and write files with the [File](#) APIs.

A [File](#) object is suited to reading or writing large amounts of data in start-to-finish order without skipping around.

All Android devices have two file storage areas: "internal" and "external" storage. These names come from the early days of Android, when most devices offered built-in non-volatile memory (internal storage), plus a removable storage medium such as a micro SD card (external storage). Some devices divide the permanent storage space into "internal" and "external" partitions, so even without a removable storage medium, there are always two storage spaces and the API behavior is the same whether the external storage is removable or not.

When saving a file to internal storage, you can acquire the appropriate directory as a [File](#) by calling one of two methods:

[getFilesDir\(\)](#)

Returns a [File](#) representing an internal directory for your app.

[getCacheDir\(\)](#)

Returns a [File](#) representing an internal directory for your app's temporary cache files. Be sure to delete each file once it is no longer needed and implement a reasonable size limit for the amount of memory you use at any given time, such as 1MB. If the system begins running low on storage, it may delete your cache files without warning.

To create a new file in one of these directories, you can use the [File\(\)](#) constructor, passing the [File](#) provided by one of the above methods that specifies your internal storage directory. For example:

```
File file = new File(context.getFilesDir(), filename);
```

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.

Although the external storage is modifiable by the user and other apps, there are two categories of files you might save here:

Public files

Files that should be freely available to other apps and to the user. When the user uninstalls your app, these files should remain available to the user.

For example, photos captured by your app or other downloaded files.

Private files

Files that rightfully belong to your app and should be deleted when the user uninstalls your app. Although these files are technically accessible by the user and other apps because they are on the external storage, they are files that realistically don't provide value to the user outside your app. When the user uninstalls your app, the system deletes all files in your app's external private directory.

For example, additional resources downloaded by your app or temporary media files.

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`](#).

Example

Layout

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"

    android:paddingLeft="@dimen/activity_horizontal_margin"

    android:paddingRight="@dimen/activity_horizontal_margin"
        android:paddingTop="@dimen/activity_vertical_margin"

    android:paddingBottom="@dimen/activity_vertical_margin"
        android:orientation="vertical"
        android:weightSum="1"
        tools:context=".MainActivity">
    <TextView
        android:id="@+id/writable"
        android:text="@string/writable"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <TextView
        android:id="@+id/readable"
        android:text="@string/readable"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <EditText
```

```

        android:id="@+id/filename"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="filename.txt"/>
<EditText
    android:id="@+id/text"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"
    android:inputType="textMultiLine"
    android:gravity="start"
    android:text="type text here"/>
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:weightSum="2"
    android:orientation="horizontal">
    <Button
        android:layout_width="0dp"
        android:layout_weight="1"
        android:layout_height="wrap_content"
        android:text="Save"
        android:onClick="Save"/>
    <Button android:layout_width="0dp"
        android:layout_weight="1"
        android:layout_height="wrap_content"
        android:text="Load"
        android:onClick="Load"/>
</LinearLayout>
</LinearLayout>

```

Strings

```

<resources>
    <string name="app_name">SavingFileOnExternalStorage</string>
    <string name="writable">Writable: </string>
    <string name="readable">Readable: </string>
    <string name="action_settings">Settings</string>
</resources>

```

Create class fields:

```

private static final String LOG_TAG = "file";
static final int READ_BLOCK_SIZE = 100;
private TextView mWritable;
private TextView mReadable;
private TextView mFileName;
private TextView mText;

```

Initialize them in onCreate() method:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mWritable = (TextView) findViewById(R.id.writable);
    if(isExternalStorageWritable() == true)
        mWritable.setText("Writable: yes");
    else
        mWritable.setText("Writable: no");
    mReadable = (TextView) findViewById(R.id.readable);
    if(isExternalStorageReadable() == true)
        mReadable.setText("Readable: yes");
    else
        mReadable.setText("Readable: no");
    mFileName = (TextView) findViewById(R.id.filename);
    mText = (TextView) findViewById(R.id.text);
}
```

Add the following methods to your Activity class:

```
// Checks if external storage is available to at least read
public boolean isExternalStorageReadable() {
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state) ||
        Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
        return true;
    }
    return false;
}
```

```
// 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;
}
```

Add a class method which returns a path to a file:

```

// A method that creates a directory in the public directory. Public
// directory
// may be of: DIRECTORY_ALARMS, DIRECTORY_DCIM, DIRECTORY_DOCUMENTS,
// DIRECTORY_DOWNLOADS, DIRECTORY_MOVIES, DIRECTORY_MUSIC,
// DIRECTORY_NOTIFICATIONS, DIRECTORY_PICTURES, DIRECTORY_PODCASTS,
// DIRECTORY_RINGTONES
public File getAlbumStorageDir(String albumName) {
    // Get the directory for the user's public directory
    //File file = new
File(Environment.getExternalStoragePublicDirectory(
    // Environment.DIRECTORY_DOCUMENTS), albumName);
    // getting the a'la "SD Card" directory
    File file = new File(Environment.getExternalStorageDirectory(),
albumName);
    if (!file.mkdirs()) {
        Log.e(LOG_TAG, "Directory not created");
    }
    return file;
}

```

Add Save() and Load() methods:

```

public void Save(View view) throws IOException {
    File fileDir = getAlbumStorageDir("test");
    String strNewFileName = mFileName.getText().toString();
    String strFileContents = mText.getText().toString();
    File newFile = new File(fileDir, strNewFileName);
    newFile.createNewFile();
    FileOutputStream fo = new
FileOutputStream(newFile.getAbsolutePath());
    fo.write(strFileContents.getBytes());
    fo.close();
}

```

```

public void Load(View view) {
    try {
        String strFileName = mFileName.getText().toString();
        File fileDir = getAlbumStorageDir("test");
        File newFile = new File(fileDir, strFileName);
        FileInputStream fileIn = new FileInputStream(newFile);
        InputStreamReader InputRead = new InputStreamReader(fileIn);
        char[] inputBuffer = new char[READ_BLOCK_SIZE];
        String s = "";
        int charRead;
        while ((charRead = InputRead.read(inputBuffer)) > 0) {
            // char to string conversion
            String readstring = String.copyValueOf(inputBuffer, 0,
charRead);

```

```

        s += readstring;
    }
    mText.setText(s);
    InputRead.close();
} catch (Exception e) {
    e.printStackTrace();
}
}

```

Edit (don't copy) AndroidManifest file, add a permission:

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.savingfiletoexternalstorage.app" >
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" /
<
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>

```

