1 What is Android external storage?

What Android "external storage" means is described in Android SDK Document:

> Every Android-compatible device supports a shared "external storage" that you can use to save files. This can be a removable storage media (such as an SD card) or an internal (non-removable) storage. Files saved to the external storage are world-readable and can be modified by the user when they enable USB mass storage to transfer files on a computer.

So don't be confused by the word "external" here. External storage can better be thought as media or shared storage. Traditionally this is an removable SD (Secure Digital) card, but it may also be implemented as built-in non-removable storage in a device that is distinct from the protected internal storage and can be mounted as a filesystem on a computer.

2 Why external storage?

Android has already provided efficient internal storage for application, but still there is much need for external storage under certain circumstance.

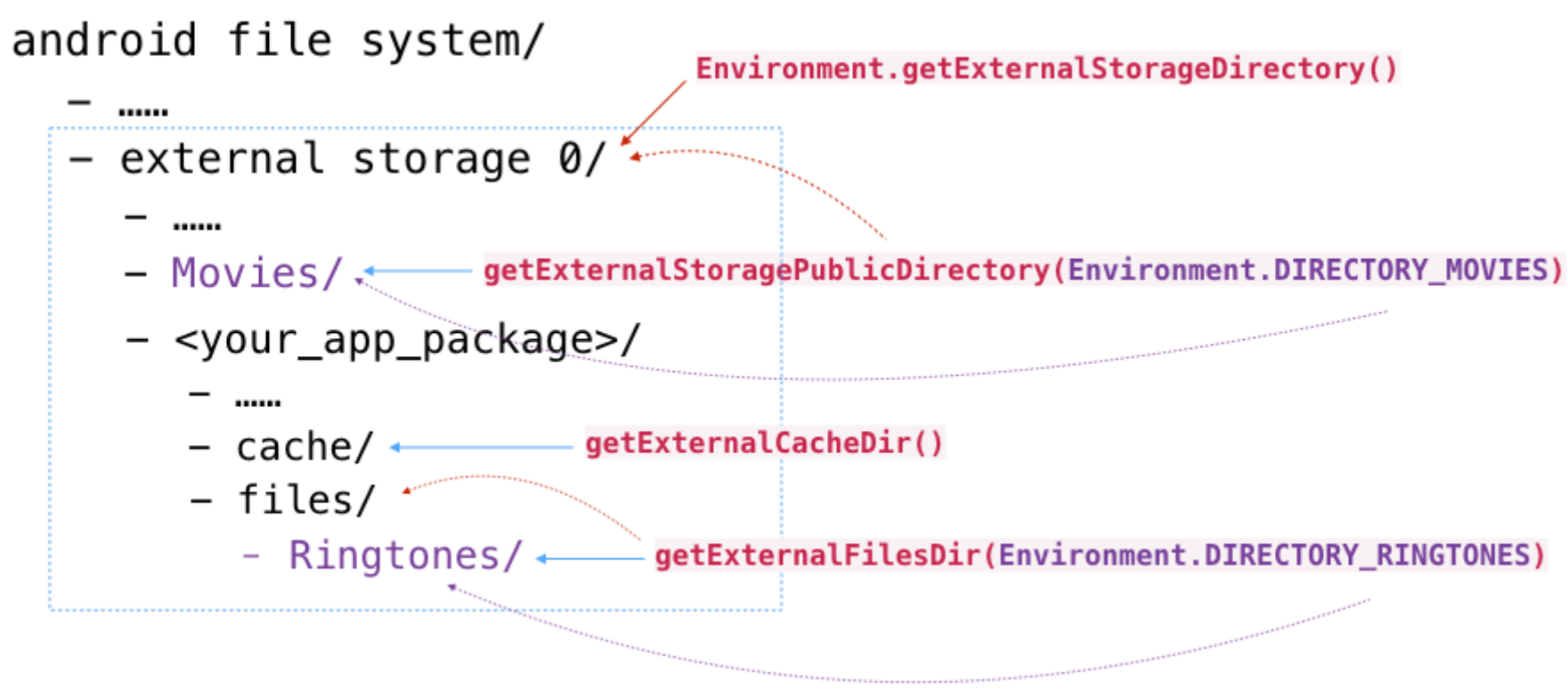
1. Need more memory or disk space to save big files;
2. Let data saved or generated in your application be accessed by other applications;
3. Some saved data should not be deleted although your application is uninstalled. For example, pictures, videos downloaded by your application.

3 Android external storage APIs Overview

Main APIs for Android external storage.

- `Environment.getExternalStorageDirectory()` : return the **primary** external storage root directory.
- `Context.getExternalFilesDir(String type)` : return the absolute path of the directory on the **primary** external storage where the application can place its own files.
- `Context.getExternalCacheDir()` | return reference to your application specific path of cache directory on external storage.
- `Environment.getExternalStoragePublicDirectory(String type)` | return public external storage directory for saving files of a particular type.

Following figure gives an overview of Android external storage APIs.



3.1 Environment.getExternalStorageDirectory()

`Environment.getExternalStorageDirectory()` returns top-level directory of the primary external storage.

If device has multiple external storage directories, returned directory represents the **primary** external storage that the user will interact with. There is also APIs available for accessing secondary storage or getting external storage directories list.

- `Context.getExternalFilesDirs(String type)`
- `Context.getExternalCacheDirs()`
- `Context.getExternalMediaDirs()`

It is noticed that the returned directory of `Environment.getExternalStorageDirectory()` is the top-level directory of the external storage. You application should avoid placing files directly under this top-level directory. If your application needs save public or shared data, you'd better use directory returned by `getExternalStoragePublicDirectory(String type)` ; on the other hand, if your application only needs to store its own internal data on external storage, you'd better consider using `getExternalFilesDir(String)` or `getExternalCacheDir()` instead.

3.2 Context.getExternalFilesDir(String type)

Returns the absolute path to the directory on the primary shared or external storage device where the application can place persistent files it owns. These files are internal to the application.

The returned directory is owned by the application and its contents will be deleted when the applicaiton is uninstalled.

The `type` parameter can be `null` or one of the following constant value.

- `Environment.DIRECTORY_MUSIC`
- `Environment.DIRECTORY_PODCASTS`
- `Environment.DIRECTORY_RINGTONES`
- `Environment.DIRECTORY_ALARMS`
- `Environment.DIRECTORY_NOTIFICATIONS`
- `Environment.DIRECTORY_PICTURES`
- `Environment.DIRECTORY_MOVIES`

If the `type` parameter is `null`, the returned path will be the root the files directory; otherwise, will be a sub directory of the given type.

3.3 Context.getExternalCacheDir()

Returns absolute path to application-specific directory on the primary shared or external storage device where the application can place cache files it owns.

Cached files under returned directory will be deleted when the application is uninstalled. Android platform does not always monitor the space available in shared storage, and thus may not automatically delete these cached files. Your application itself should always manage the maximum space used in this location.

3.4 Environment.getExternalStoragePublicDirectory(String type)

Get a top-level shared or external storage directory for placing files of a particular type.

The `type` parameter CAN NOT be `null`, should be one of the following constant value.

- `Environment.DIRECTORY_MUSIC`
- `Environment.DIRECTORY_PODCASTS`
- `Environment.DIRECTORY_RINGTONES`
- `Environment.DIRECTORY_ALARMS`
- `Environment.DIRECTORY_NOTIFICATIONS`
- `Environment.DIRECTORY_PICTURES`
- `Environment.DIRECTORY_MOVIES`

Because this returned directory is public and is for shared files, you application should be careful and avoid erasing any files here.

4 How to Use Android external storage

External storage of Android device may not always be available, for example:

- external storage may not be accessible if it has been mounted by users on their computer;
- external storage has been removed from device.

So the first step is to check state of the external storage. External storage state can be checked using `Environment.getExternalStorageState(File path)`. More details can refer to Android Tutorial: Check SD Card Status.

The second step is to add `android.permission.READ_EXTERNAL_STORAGE` or `android.permission.WRITE_EXTERNAL_STORAGE` permission to your application.

Note: From Android 6.0+, application has to ask user for a permission one-by-one at runtime instead of being granted any permission at installation time.

The third step is to get `File` object reference of external storage directory.

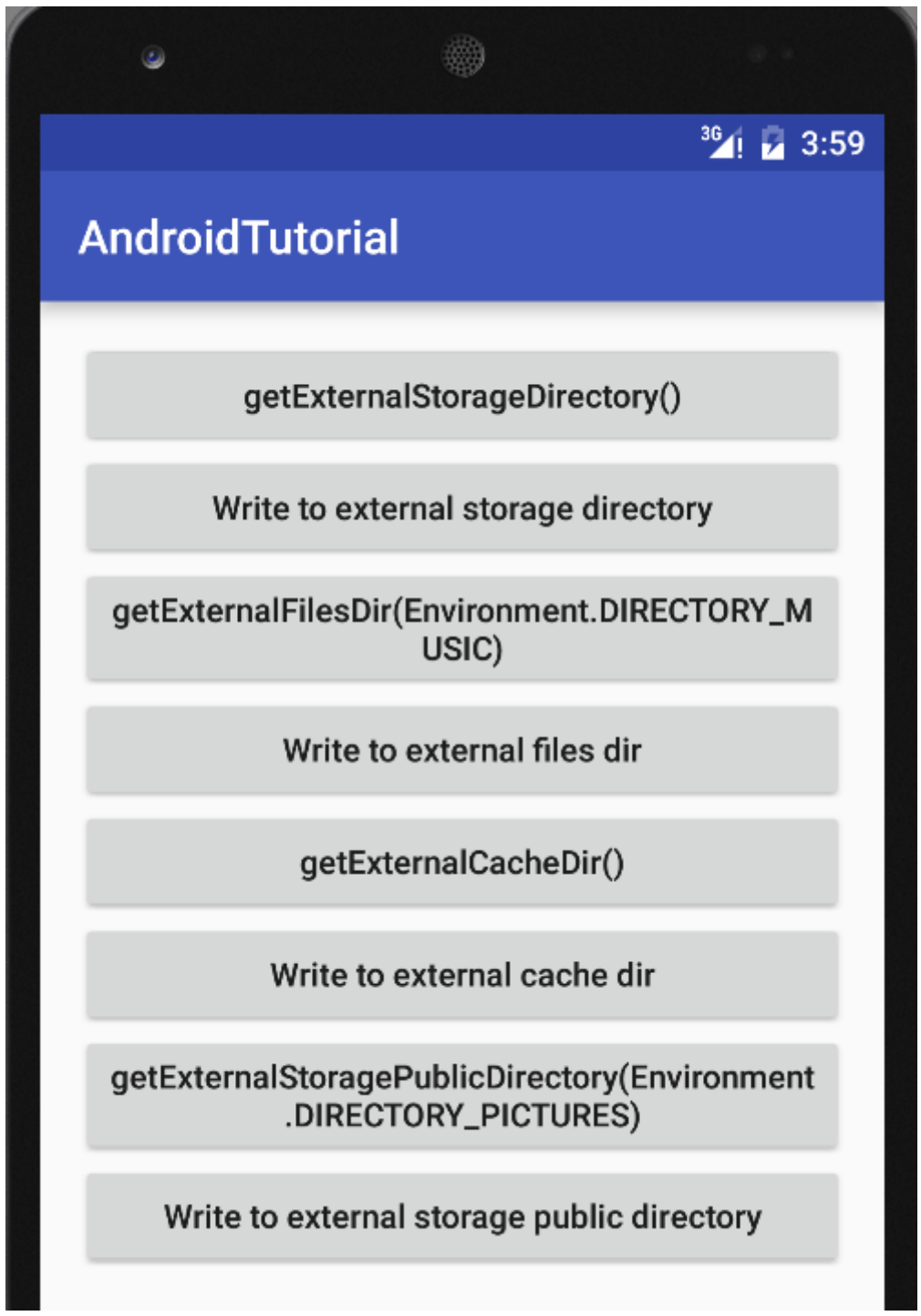
Lastly, write or read data using common `java.io` APIs with the directory `File` object.

5 Android external storage example

There are two main features in this simple demo:

- Show various external storage full path using `Toast`;
- Try to write demo file to external storage;

Main UI of the demo looks like screenshot below.



5.1 Add `WRITE_EXTERNAL_STORAGE` permission

Add `android.permission.WRITE_EXTERNAL_STORAGE` permission to `AndroidManifest.xml` file of your Android application project.

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

5.2 Helper method to check external storage state

Create a helper method in `Activity` class to check external storage state.

```
private boolean checkSDCardStatus() {
    String SDCardStatus = Environment.getExternalStorageState();

    // MEDIA_UNKNOWN: unrecognized SD card
    // MEDIA_REMOVED: no SD card at all
    // MEDIA_UNMOUNTED: SD card exist but not mounted, not available in Android 4.0+
    // MEDIA_CHECKING: preparing SD card, e.g. powered on and booting
    // MEDIA_MOUNTED: mounted and ready to use
    // MEDIA_MOUNTED_READ_ONLY
    switch (SDCardStatus) {
        case Environment.MEDIA_MOUNTED:
            return true;
        case Environment.MEDIA_MOUNTED_READ_ONLY:
            Toast.makeText(this, "SD card is ready only.", Toast.LENGTH_LONG).show();
            return false;
        default:
            Toast.makeText(this, "SD card is not available.", Toast.LENGTH_LONG).show();
            return false;
    }
}
```

5.3 Helper method to write file

Create a common helper method to write string to a file.

```
private void writeDataToPath(File path, String fileName, String data) {
    File targetFilePath = new File(path, fileName);
    FileOutputStream fos = null;

    try {
        fos = new FileOutputStream(targetFilePath);
        fos.write(data.getBytes());
    } catch (Exception e) {
        e.printStackTrace();
        Toast.makeText(this, "Failed: " + e.getMessage(), Toast.LENGTH_LONG).show();
    } finally {
        if (fos != null) {
            try {
                Toast.makeText(this, "Write to <" + targetFilePath.getAbsolutePath() + "> successfully!", Toast.LENGTH_LONG).show();
                fos.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        } else {
            Toast.makeText(this, "Failed to write!", Toast.LENGTH_LONG).show();
        }
    }
}
```

5.4 Listen button clicking

Lastly, implement `View.OnClickListener` interface in the demo Activity.

```
@Override
public void onClick(View v) {
    if (! this.checkSDCardStatus()) {
        return;
    }

    File path = null;
    String fileName = null;
    String contentData = null;

    switch (v.getId()) {
        case R.id.btn_get_external_storage_directory:
            path = Environment.getExternalStorageDirectory();
            Toast.makeText(this, path.getAbsolutePath(), Toast.LENGTH_LONG).show();
            break;
        case R.id.btn_write_to_external_storage_directory:
            path = Environment.getExternalStorageDirectory();
            fileName = "getExternalStorageDirectory.txt";
            contentData = "getExternalStorageDirectory() demo";
            this.writeDataToPath(path, fileName, contentData);
            break;
        case R.id.btn_get_external_files_dir:
            path = getExternalFilesDir(Environment.DIRECTORY_MUSIC);
            Toast.makeText(this, path.getAbsolutePath(), Toast.LENGTH_LONG).show();
            break;
        case R.id.btn_write_to_external_files_dir:
            path = getExternalFilesDir(Environment.DIRECTORY_MUSIC);
            fileName = "getExternalFilesDir.txt";
            contentData = "getExternalFilesDir(Environment.DIRECTORY_MUSIC) demo";
            this.writeDataToPath(path, fileName, contentData);
            break;
        case R.id.btn_get_external_cache_dir:
            path = getExternalCacheDir();
            Toast.makeText(this, path.getAbsolutePath(), Toast.LENGTH_LONG).show();
            break;
        case R.id.btn_write_to_external_cache_dir:
            path = getExternalCacheDir();
            fileName = "getExternalCacheDir.txt";
            contentData = "getExternalCacheDir() demo";
            this.writeDataToPath(path, fileName, contentData);
            break;
        case R.id.btn_get_external_storage_public_directory:
            path = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES);
            Toast.makeText(this, path.getAbsolutePath(), Toast.LENGTH_LONG).show();
            break;
        case R.id.btn_write_to_external_storage_public_directory:
            path = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES);
            fileName = "getExternalStoragePublicDirectory.txt";
            contentData = "getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES) demo";
            this.writeDataToPath(path, fileName, contentData);
            break;
        default:
```

```
            break;
        }
    }
```

You can check the saved file via file explore of Android Device Monitor (DDMS) in Android Studio or Eclipse.