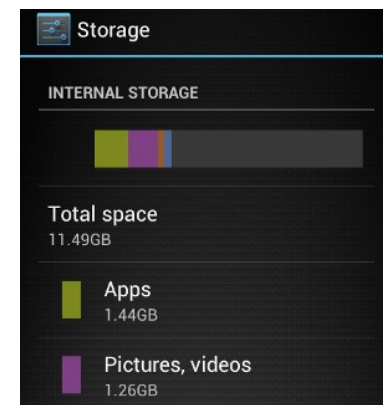


Files and storage

- Android can read/write files from two locations:
 - **internal** and **external** storage.
 - Both are **persistent** storage; data remains after power-off / reboot.
- **internal storage**: Built into the device.
 - guaranteed to be present
 - typically smaller
 - can't be expanded or removed
 - specific and private to each app
 - wiped out when the app is uninstalled



File ([link](#)) and Streams ([link](#))

- **java.io.File** - Objects that represent a file or directory.
 - methods: `canRead`, `canWrite`, `create`, `delete`, `exists`, `getName`, `getParent`, `getPath`, `isFile`, `isDirectory`, `lastModified`, `length`, `listFiles`, `mkdir`, `mkdirs`, `renameTo`
- **java.io.InputStream**, **OutputStream** - Stream objects represent flows of data bytes from/to a source or destination.
 - Could come from a file, network, database, memory, ...
 - Normally not directly used; they only include low-level methods for reading/writing a byte (character) at a time from the input.
 - Instead, a stream is often passed as parameter to other objects like **java.util.Scanner**, **java.io.BufferedReader**, **java.io.PrintStream** to do the actual reading / writing.

Using internal storage ([link](#))

- An activity has methods you can call to read/write files:
 - `getFilesDir()` - returns internal directory for your app
 - `getCacheDir()` - returns a "temp" directory for scrap files
 - `getResources().openRawResource(R.raw.id)`
 - read an input file from `res/raw/`
 - `openFileInput("name", mode)` - opens a file for reading
 - `openFileOutput("name", mode)` - opens a file for writing
- You can use these to read/write files on the device.
 - many methods return standard `java.io.File` objects
 - some return `java.io.InputStream` or `OutputStream` objects, which can be used with standard classes like `Scanner`, `BufferedReader`, and `PrintStream` to read/write files (see Java API)

Internal storage example 1

```
// read a file, and put its contents into a TextView
// (assumes hello.txt file exists in res/raw/ directory)
Scanner scan = new Scanner(
    getResources().openRawResource(R.raw.hello));
String allText = "";    // read entire file
while (scan.hasNextLine()) {
    String line = scan.nextLine();
    allText += line;
}
myTextView.setText(allText);
scan.close();
```

Internal storage example 2

```
// write a short text file to the internal storage
PrintStream output = new PrintStream
    (openFileOutput("out.txt", MODE_PRIVATE));
output.println("Hello, world!");
output.println("How are you?");
output.close();
...
// read the same file, and put its contents into a TextView
Scanner scan = new Scanner(
    openFileInput("out.txt", MODE_PRIVATE));
String allText = "";    // read entire file
while (scan.hasNextLine()) {
    String line = scan.nextLine();
    allText += line;
}
myTextView.setText(allText);
scan.close();
```

External storage

- **external storage:** Card that is inserted into the device.
(*such as a MicroSD card*)
 - can be much larger than internal storage
 - can be removed or transferred to another device if needed
 - may not be present, depending on the device
 - read/writable by other apps and users; not private to your app
 - *not* wiped when the app is uninstalled, except in certain cases



External storage permission

- If your app needs to read/write the device's external storage, you must explicitly request **permission** to do so in your app's **AndroidManifest.xml** file.
 - On install, the user will be prompted to confirm your app permissions.

```
<manifest ...>
```

```
  <uses-permission
```

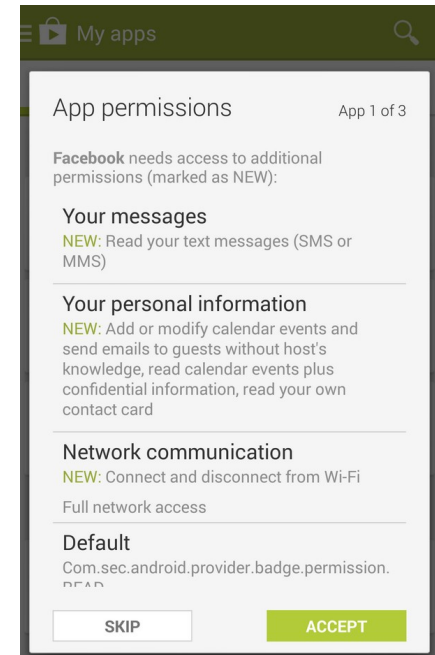
```
    android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

```
  <uses-permission
```

```
    android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

```
  ...
```

```
</manifest>
```



Using external storage

- Methods to read/write external storage:
 - `getExternalFilesDir("name")` - returns "private" external directory for your app with the given name
 - `Environment.getExternalStoragePublicDirectory(name)` - returns public directory for common files like photos, music, etc.
 - pass constants for **name** such as `Environment.DIRECTORY_ALARMS`, `DIRECTORY_DCIM`, `DIRECTORY_DOWNLOADS`, `DIRECTORY_MOVIES`, `DIRECTORY_MUSIC`, `DIRECTORY_NOTIFICATIONS`, `DIRECTORY_PICTURES`, `DIRECTORY_PODCASTS`, `DIRECTORY_RINGTONES`
- You can use these to read/write files on the external storage.
 - the above methods return standard `java.io.File` objects
 - these can be used with standard classes like `Scanner`, `BufferedReader`, and `PrintStream` to read/write files (see Java API)

External storage example

```
// write short data to app-specific external storage
File outDir = getExternalFilesDir(null);    // root dir
File outFile = new File(outDir, "example.txt");
PrintStream output = new PrintStream(outFile);
output.println("Hello, world!");
output.close();
```

```
// read list of pictures in external storage
File picsDir =
    Environment.getExternalStoragePublicDirectory(
        Environment.DIRECTORY_PICTURES);
for (File file : picsDir.listFiles()) {
    ...
}
// to read from external:
File inDir = getExternalFilesDir(null);
File inFile = new File(inDir, "name.txt");
Scanner output = new Scanner(inFile);
String allText="";
while(output.hasNextLine()){
    String x= output.nextLine();
    allText+=x+"\n";
}
```

Checking if storage is available

```
/* Checks if external storage is available  
 * for reading and writing */
```

```
public boolean isExternalStorageWritable() {  
    return Environment.MEDIA_MOUNTED.equals(  
        Environment.getExternalStorageState());  
}
```

```
/* Checks if external storage is available  
 * for reading */
```

```
public boolean isExternalStorageReadable() {  
    return isExternalStorageWritable() ||  
        Environment.MEDIA_MOUNTED_READ_ONLY.equals(  
            Environment.getExternalStorageState());  
}
```

Accessing web data ([link](#))

- To read data from the web, first request the INTERNET permission in your **AndroidManifest.xml**:

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

- Then you can use the standard `java.net.URL` class to connect to a file or page at a given URL and read its data:

```
URL url = new URL("http://foobar.com/example.txt");  
Scanner scan = new Scanner(url.openStream());  
while (scan.hasNextLine()) {  
    String line = scan.nextLine();  
    ...  
}
```