

1 SAVING FILES ON INTERNAL STORAGE

Sources:

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

<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);
```

Alternatively, you can call [openFileOutput\(\)](#) to get a [FileOutputStream](#) that writes to a file in your internal directory.

Example 1

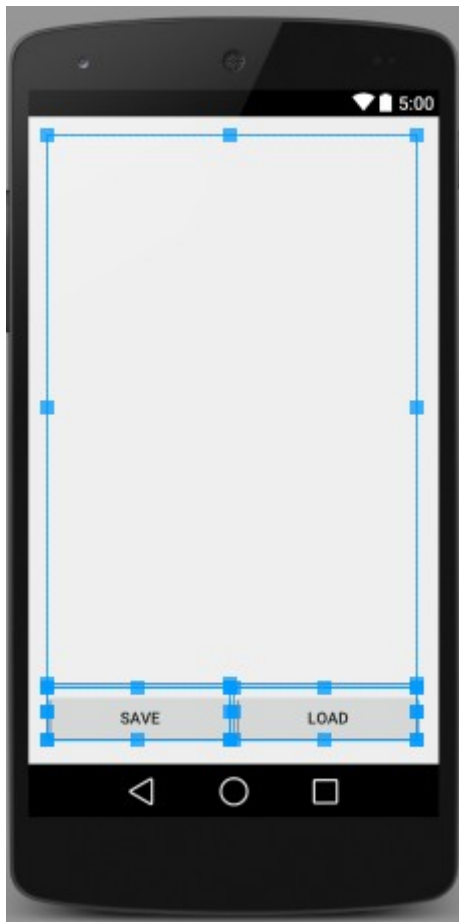
Layout

```
<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">
    <EditText
        android:id="@+id/text"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:gravity="top"
        android:inputType="textMultiLine"
    android:layout_weight="1"/>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:weightSum="2">
        <Button
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="@string/bWriteText"
            android:onClick="onSave"/>
        <Button
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="@string/bReadText"
            android:onClick="onLoad"/>
    </LinearLayout>
</LinearLayout>
```



Strings:

```
<resources>
    <string name="app_name">InternalStorageFiles</string>
    <string name="action_settings">Settings</string>
    <string name="bReadText">Load</string>
    <string name="bWriteText">Save</string>
</resources>
```

Add class field and instantiate it in onCreate() method:

```
private EditText mEditText;
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mEditText = (EditText) findViewById(R.id.text);
}
```

Create public void onSave(View view) and public void onLoad(View view) methods:

```
public void onSave(View view) {
    try {
        FileOutputStream fileout = openFileOutput("mytextfile.txt",
        MODE_PRIVATE);
        DataOutputStream dataOutStr = new DataOutputStream(fileout);
        dataOutStr.writeBytes(mEditText.getText().toString());
        dataOutStr.close();
        fileout.close();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}
```

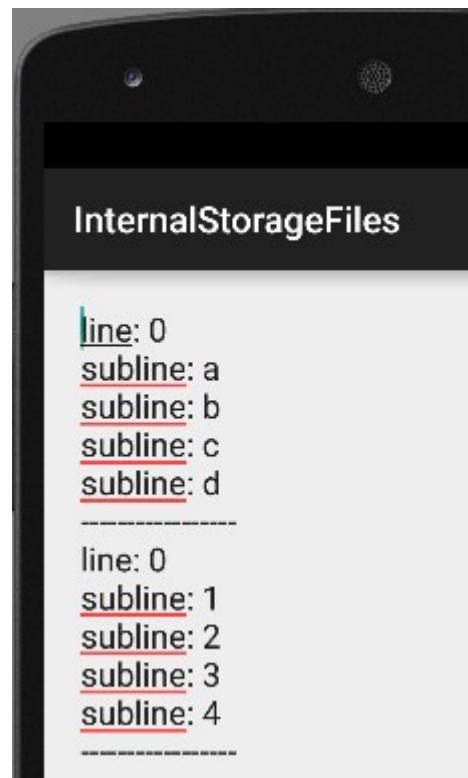
```
public void onLoad(View view) {
    try{
        FileInputStream fileIn = openFileInput("mytextfile.txt");
        BufferedReader br = new BufferedReader(new
        InputStreamReader(fileIn));
        String line = null;
        Boolean theFirstLine = true;
        StringBuilder strBuilder = new StringBuilder();
        while((line = br.readLine())!= null ){
            if(theFirstLine != true) strBuilder.append("\n");
            else theFirstLine = false;
            strBuilder.append(line);
        }
        mEditText.setText(strBuilder.toString());
        br.close();
        fileIn.close();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}
```

Run the application, type a text, save, relaunch the application and load the file.
To split a line into sublines, use regular expression:

```
// \\s+ means any number of whitespaces between tokens
String [] subline = line.split("\\s+");
String var_1 = subline[0];
String var_2 = subline[1];
...
```

Change void onLoad(View view) to:

```
public void onLoad(View view) {
    try{
        FileInputStream fileIn = openFileInput("mytextfile.txt");
        BufferedReader br = new BufferedReader(new
InputStreamReader(fileIn));
        String line = null;
        Boolean theFirstLine = true;
        StringBuilder strBuilder = new StringBuilder();
        int i = 0;
        while((line = br.readLine())!= null ){
            if(theFirstLine != true) strBuilder.append("\n");
            else theFirstLine = false;
            // \s+ means any number of whitespaces between tokens
            String [] subline = line.split("\\s+");
            strBuilder.append("line: " + i++ + "\n");
            for(String subL : subline)
                strBuilder.append("subline: " + subL + "\n");
            strBuilder.append("-----");
        }
        mEditText.setText(strBuilder.toString());
        br.close();
        fileIn.close();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}
```



Convert, if needed, String to int, float, double etc:

```
int x = Integer.parseInt(String string);
float x = Float.parseFloat(String string);
double x = Double.parseDouble(String string);
boolean x = Boolean.parseBoolean(String string);
...
```

DataOutputStream allows writing to file not only String, but other types too:

```
dataOutStr.writeInt(int val);
dataOutStr.writeFloat(Float val);
dataOutStr.writeFouble(double val);
dataOutStr.writeLong(long val);
dataOutStr.writeShort(short val);
dataOutStr.writeBoolean(boolean val);
```

etc.

Example 2

Create a few EditText fields in your layout file, get their strings, convert to appropriate types and save to a file:

```
public void onSave(View view) {
    try {
        FileOutputStream fileout = openFileOutput("mytextfile.txt",
        MODE_PRIVATE);
        DataOutputStream dataOutStr = new DataOutputStream(fileout);

        dataOutStr.writeInt(1);
        dataOutStr.writeDouble(2.2);

        dataOutStr.close();
        fileout.close();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}
```

```
public void onLoad(View view) {
    try{
        FileInputStream fileIn = openFileInput("mytextfile.txt");
        DataInputStream dis = new DataInputStream(fileIn);
        int i = dis.readInt();
        double d = dis.readDouble();
        mEditText.setText("Integer: " + i + "\ndouble: " + d);
        dis.close();
        fileIn.close();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}
```