Session 8 Android Studio

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

How to make a screenshot and transfer this file to directory C:/a

1. Open some app , so you can distinguish the screenshot

adb -d shell screencap -p /sdcard/vasya.png (or any other image format like .jpg, jpeg, bmp, etc)

2. check if vasya.png is in sdcard

adb –d shell

ls

cd sdcard

ls (to check content)

check for file

3. copy vasya.png to C:/a/

adb -d pull /sdcard/vasya.png C:/a/

check Directory C:/a/ if vasya.png is there

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

How to record a 1 min video and move this file to C:/a/?

this command only accepts seconds (not minute listing)

Most common Video formats are : .mpeg, .mpeg4, avi...etc

1. adb -d shell screenrecord --time-limit 60 /sdcard/happy.mp4

2. check if happy.mp4 is in sdcard
adb -d shell
ls
cd sdcard
ls (to check content and look up file)

3. copy happy.mp4 to C:/a/
adb -d pull /sdcard/ happy.mp4 C:/a/
check Directory C:/a/ if happy.mp4 is there

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

How to create directory on your Real Device

adb -d shell

ls (if you wish to create sub-directory in already existing , exp sdcard)
cd sdcard
mkdir Natalia

if not subdirectory needed , then
adb -d shell
mkdir Natalia

1. Create sub-directory in sdcard (sdcard/Natalia)
ls
cd sdcard
mkdir Natalia

exit shell (type exit)

2. Copy fish.txt to sdcard/Natalia
adb -d push C:/a/fish.txt /sdcard/Natalia

3. Find fish.txt using ls
adb -d shell
ls
cd sdcard

cd Natalia

How to remove sub-directory (if it does have files, then you have to remove files first , then remove sub-directory)

adb -d shell
ls
cd sdcard
cd Natalia
rm fish.txt
rmdir Natalia

or you may just use :

adb -d shell
ls
cd sdcard
rmdir –r Natalia (older Android Studio)

rm –r Natalia (current 3.1.2)

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

How to shut down and remove your emulator

adb -s emulator-5554 emu kill or adb –e emu kill
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

type adb devices
should see real device only

No more emulator-5554 (this means no more -d - no separation in the command line)

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Reboot the Real Device

adb reboot

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

What is input keyevent ??????
You may manipulate with controls on your device
https://developer.android.com/reference/android/view/KeyEvent.html
or check another link below

Play around ☺
adb shell

input keyevent 26 (power off/on)
input keyevent 82 (unlock your screen)

https://stackoverflow.com/questions/7789826/adb-shell-input-events

0 --> "KEYCODE_UNKNOWN"
1 --> "KEYCODE_MENU"
2 --> "KEYCODE_SOFT_RIGHT"
3 --> "KEYCODE_HOME"
4 --> "KEYCODE_BACK"
5 --> "KEYCODE_CALL"
6 --> "KEYCODE_ENDCALL"
7 --> "KEYCODE_0"
8 --> "KEYCODE_1"
9 --> "KEYCODE_2"
10 --> "KEYCODE_3"
11 --> "KEYCODE_4"
12 --> "KEYCODE_5"
13 --> "KEYCODE_6"
14 --> "KEYCODE_7"
15 --> "KEYCODE_8"
16 --> "KEYCODE_9"

17 --> "KEYCODE_STAR"
18 --> "KEYCODE_POUND"
19 --> "KEYCODE_DPAD_UP"
20 --> "KEYCODE_DPAD_DOWN"
21 --> "KEYCODE_DPAD_LEFT"
22 --> "KEYCODE_DPAD_RIGHT"
23 --> "KEYCODE_DPAD_CENTER"
24 --> "KEYCODE_VOLUME_UP"
25 --> "KEYCODE_VOLUME_DOWN"
26 --> "KEYCODE_POWER"
27 --> "KEYCODE_CAMERA"
28 --> "KEYCODE_CLEAR"
29 --> "KEYCODE_A"
30 --> "KEYCODE_B"
31 --> "KEYCODE_C"
32 --> "KEYCODE_D"
33 --> "KEYCODE_E"
34 --> "KEYCODE_F"
35 --> "KEYCODE_G"
36 --> "KEYCODE_H"
37 --> "KEYCODE_I"
38 --> "KEYCODE_J"

39 --> "KEYCODE_K"
40 --> "KEYCODE_L"
41 --> "KEYCODE_M"
42 --> "KEYCODE_N"
43 --> "KEYCODE_O"
44 --> "KEYCODE_P"
45 --> "KEYCODE_Q"
46 --> "KEYCODE_R"
47 --> "KEYCODE_S"
48 --> "KEYCODE_T"
49 --> "KEYCODE_U"
50 --> "KEYCODE_V"
51 --> "KEYCODE_W"
52 --> "KEYCODE_X"
53 --> "KEYCODE_Y"
54 --> "KEYCODE_Z"
55 --> "KEYCODE_COMMA"
56 --> "KEYCODE_PERIOD"
57 --> "KEYCODE_ALT_LEFT"
58 --> "KEYCODE_ALT_RIGHT"
59 --> "KEYCODE_SHIFT_LEFT"
60 --> "KEYCODE_SHIFT_RIGHT"

61 --> "KEYCODE_TAB"
62 --> "KEYCODE_SPACE"
63 --> "KEYCODE_SYM"
64 --> "KEYCODE_EXPLORER"
65 --> "KEYCODE_ENVELOPE"
66 --> "KEYCODE_ENTER"
67 --> "KEYCODE_DEL"
68 --> "KEYCODE_GRAVE"
69 --> "KEYCODE_MINUS"
70 --> "KEYCODE_EQUALS"
71 --> "KEYCODE_LEFT_BRACKET"
72 --> "KEYCODE_RIGHT_BRACKET"
73 --> "KEYCODE_BACKSLASH"
74 --> "KEYCODE_SEMICOLON"
75 --> "KEYCODE_APOSTROPHE"
76 --> "KEYCODE_SLASH"
77 --> "KEYCODE_AT"
78 --> "KEYCODE_NUM"
79 --> "KEYCODE_HEADSETHOOK"
80 --> "KEYCODE_FOCUS"
81 --> "KEYCODE_PLUS"
82 --> "KEYCODE_MENU"

83 --> "KEYCODE_NOTIFICATION"
84 --> "KEYCODE_SEARCH"
85 --> "TAG_LAST_KEYCODE"

Dumpsys Commands

https://stackoverflow.com/questions/11201659/whats-the-android-adb-shell-dumpsys-tool-and-what-are-its-benefits

dumpsys is a tool that runs on Android devices and provides information about system services.

You can call **dumpsys** from the command line using the Android Debug Bridge (ADB) to get diagnostic output for all system services running on a connected device.

(Please note : Some of the services might not work on your device)

For a complete list of system services that you can use with **dumpsys**, use the following command:

```
adb shell dumpsys -l
```

the command below provides system data for input components, such as *touchscreens or built-in keyboards*:

```
adb shell dumpsys input
```

Test UI performance

Specifying the **gfxinfo** service provides output with performance information relating to frames of animation that are occurring during the recording phase. The following command uses **gfxinfo** to gather UI performance data for a specified package name:

You have to have your app open in order to run this command properly. We are using United APP

```
adb shell dumpsys gfxinfo com.united.mobile.android
```

To find the UID for your app, run this command:

We are using our United APP that has a package name: com.united.mobile.android

adb shell dumpsys package *com.united.mobile.android*

Then look for the line labeled `userId`.

For example, to find network usage for the app 'com.example.myapp', run the following command:

adb shell dumpsys package *com.united.mobile.android* | grep userId

if the above command line is not recognizing "grep" :

adb shell

dumpsys package *com.united.mobile.android* | grep userId ***(make sure in ID: I- is cap. d-lowercase)***

Output should be similar to the following: **userId=10007 gids=[3003, 1028, 1015]**

Using the sample dump above, look for lines that have `uid=10007`.

Two such lines exist—the first indicates a mobile connection and the second indicates a Wi-Fi connection.

This is an Output example that I will be using:

```
  ident=[[type=WIFI, subType=COMBINED, networkId="MySSID"]] uid=10007  set=DEFAULT tag=0x0

    NetworkStatsHistory: bucketDuration=7200000

      bucketStart=1406138400000 activeTime=7200000 rxBytes=17086802 rxPackets=15387 txBytes=1214969 txPackets=8036
operations=28
```

Below each line, you can see the following information for each two-hour window (which `bucketDuration` specifies in milliseconds):

- `set=DEFAULT` indicates foreground network usage, while `set=BACKGROUND` indicates background usage. `set=ALL` implies both.
- `tag=0x0` indicates the socket tag associated with the traffic.
- `rxBytes` and `rxPackets` represent received bytes and received packets in the corresponding time interval.

- `txBytes` and `txPackets` represent sent (transmitted) bytes and sent packets in the corresponding time interval.

Let’s check our battery

E:\AndroidStudio\android-sdk-windows\platform-tools>adb shell dumpsys **battery**

This command line will give you a complete info about your battery.

OUTPUT for battery :

Current Battery Service state:

AC powered: false

USB powered: true

Wireless powered: false

Max charging current: 500000

Max charging voltage: 5000000

Charge counter: 1938174

status: 2

health: 2

present: true

level: 65

scale: 100

voltage: 4016

temperature: 274

technology: Li-ion

To save battery info into a log file ;

E:\AndroidStudio\android-sdk-windows\platform-tools>adb shell dumpsys **battery > C:\a\battery.txt**

We can run a command line to extract only info that we need. In this example : temperature

2. E:\AndroidStudio\android-sdk-windows\platform-tools>

adb shell

/$ dumpsys battery | grep ***temperature*** (extract data . In our case its “temperature”)

xx

Inspect battery diagnostics

Specifying the `batterystats` service generates interesting statistical data about battery usage on a device, organized by unique user ID (UID).

To learn how to use `dumpsys` to test your app for Doze and App Standby, go to Testing with Doze and App Standby.

The command for `batterystats` is as follows:

```
adb shell dumpsys batterystats options  ( like:  --charged or --checkin)
```

To see a list of additional options available to `batterystats`, include the `-h` option.

The example below outputs battery usage statistics for a United app package since the device was last charged:

```
adb shell dumpsys batterystats --charged com.united.mobile.android
```

The output typically includes the following:

- History of battery-related events
- Global statistics for the device
- Approximate power use per UID and system component
- Per-app mobile milliseconds per packet
- System UID aggregated statistics
- App UID aggregated statistics

Inspecting machine-friendly output

You can generate **`batterystats`** output in machine-readable CSV format by using the following command:

```
adb shell dumpsys batterystats --checkin
```

The following is an example of the output you should see:

```
9,0,i,vers,11,116,K,L

9,0,i,uid,1000,android

9,0,i,uid,1000,com.android.providers.settings

9,0,i,uid,1000,com.android.inputdevices

9,0,i,uid,1000,com.android.server.telecom
```

Battery-usage observations may be per-UID or system-level; data is selected for inclusion based on its usefulness in analyzing battery performance. Each row represents an observation with the following elements:

- A dummy integer

- The user ID associated with the observation
- The aggregation mode:
- "i" for information not tied to charged/uncharged status.
- "l" for --charged (usage since last charge).
- "u" for --unplugged (usage since last unplugged). Deprecated in Android 5.1.1.
- Section identifier, which determines how to interpret subsequent values in the line.

meminfo

You can record a snapshot of how your app's memory is divided between different types of RAM allocation with the following command:

The -d flag prints more info related to Dalvik and ART memory usage.

Example :

```
adb shell dumpsys meminfo com.untied.mobile.android -d
```

Testing with Doze and App Standby

To ensure a great experience for your users, you should test your app fully in Doze and App Standby.

Testing your app with Doze

You can test Doze mode by following these steps:

1. Configure a hardware device or virtual device with an Android 6.0 (API level 23) or higher system image.
2. Connect the device to your development machine and install your app.
3. Run your app and leave it active.
4. Force the system into idle mode by running the following command:

```
5. $ adb shell dumpsys deviceidle force-idle
```

6. When ready, exit idle mode by running the following command:

```
7. $ adb shell dumpsys deviceidle unforce
```

8. Observe the behavior of your app after you reactivate the device. Make sure the app recovers gracefully when the device exits Doze.

Testing your app with App Standby

To test the App Standby mode with your app:

1. Configure a hardware device or virtual device with an Android 6.0 (API level 23) or higher system image.
2. Connect the device to your development machine and install your app.
3. Run your app and leave it active.
4. Force the app into App Standby mode by running the following commands:

```
5. $ adb shell dumpsys battery unplug
```

```
$ adb shell am set-inactive <packageName> true
```

6. Simulate waking your app using the following commands:

```
7. $ adb shell am set-inactive <packageName> false
```

```
$ adb shell am get-inactive <packageName>
```

8. Observe the behavior of your app after waking it. Make sure the app recovers gracefully from standby mode. In particular, you should check if your app's Notifications and background jobs continue to function as expected.

Acceptable use cases for whitelisting

The table below highlights the acceptable use cases for requesting or being on the Battery Optimizations exceptions whitelist. In general, your app should not be on the whitelist unless Doze or App Standby break the core function of the app or there is a technical reason why your app cannot use FCM high-priority messages.

For more information, see Support for other use cases .

Try dumpsys command lines and practice . Examples : cpuinfo, meminfo, wifi, location, etc

```
DUMP OF SERVICE SurfaceFlinger:
DUMP OF SERVICE accessibility:
DUMP OF SERVICE account:
DUMP OF SERVICE activity:
DUMP OF SERVICE alarm:
DUMP OF SERVICE appwidget:
DUMP OF SERVICE audio:
DUMP OF SERVICE backup:
DUMP OF SERVICE battery:
DUMP OF SERVICE batteryinfo:
DUMP OF SERVICE clipboard:
DUMP OF SERVICE connectivity:
DUMP OF SERVICE content:
DUMP OF SERVICE cpuinfo:
DUMP OF SERVICE device_policy:
DUMP OF SERVICE devicestoragemonitor:
DUMP OF SERVICE diskstats:
DUMP OF SERVICE dropbox:
DUMP OF SERVICE entropy:
DUMP OF SERVICE hardware:
DUMP OF SERVICE input_method:
DUMP OF SERVICE iphonesubinfo:
DUMP OF SERVICE isms:
DUMP OF SERVICE location:
DUMP OF SERVICE media.audio_flinger:
DUMP OF SERVICE media.audio_policy:
DUMP OF SERVICE media.player:
DUMP OF SERVICE meminfo:
DUMP OF SERVICE mount:
DUMP OF SERVICE netstat:
DUMP OF SERVICE network_management:
DUMP OF SERVICE notification:
DUMP OF SERVICE package:
DUMP OF SERVICE permission:
DUMP OF SERVICE phone:
DUMP OF SERVICE power:
DUMP OF SERVICE reboot:
```

```
DUMP OF SERVICE screenshot:
DUMP OF SERVICE search:
DUMP OF SERVICE sensor:
DUMP OF SERVICE simphonebook:
DUMP OF SERVICE statusbar:
DUMP OF SERVICE telephony.registry:
DUMP OF SERVICE throttle:
DUMP OF SERVICE usagestats:
DUMP OF SERVICE vibrator:
DUMP OF SERVICE wallpaper:
DUMP OF SERVICE wifi:
DUMP OF SERVICE window:
```

How to use Dumpsys for an application (we need to find out a package name first)

E:\AndroidStudio\android-sdk-windows\platform-tools>adb shell

OnePlus3T:/ $ pm list packages | grep calculator

package:com.oneplus.calculator (your package name has a different name)

OnePlus3T:/ $ exit

Here we go :

E:\AndroidStudio\android-sdk-windows\platform-tools>adb shell dumpsys meminfo com.oneplus.calculator