

Android Developers

dumpsys

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) (<https://developer.android.com/studio/command-line/adb.html>) to get diagnostic output for all system services running on a connected device. This output is typically more verbose than you may want, so use the command line options described below to get output for only the system services you're interested in. This page also describes how to use **dumpsys** to accomplish common tasks, such as inspecting input, RAM, battery, or network diagnostics.

Syntax

The general syntax for using **dumpsys** is as follows:

```
adb shell dumpsys [-t timeout] [--help | -l | --skip
```

To get a diagnostic output for all system services for your connected device, simply run `adb shell dumpsys`. However, this outputs far more information than you would typically want. For more manageable output, specify the service you want to examine by including it in the command. For example, the command below provides system data for input components, such as touchscreens or built-in keyboards:

```
adb shell dumpsys input
```

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

In this document

Syntax (#syntax)

Command line options
(#options)

Inspect input diagnostics (#input)

Event hub state (#event_state)

Input reader state
(#reader_state)

Input dispatcher state
(#dispatcher_state)

Things to check for
(#things_to_check_for)

Test UI performance (#ui)

Inspect network diagnostics (#network)

Active interfaces and active
UID interfaces
(#active_interfaces)

'Dev' and 'Xt' statistics
(#dev_xt)

UID stats (#uid_stats)

Inspect battery diagnostics (#battery)

Inspecting machine-friendly
output (#inspecting_machine-
friendly_output)

View memory allocations (#ViewingAllocations)

procstats (#procstats)

meminfo (#meminfo)

```
adb shell dumpsys -l
```

Command line options

The following table lists the available options when using `dumpsys`.

| Option | Description |
|----------------------------------|--|
| <code>-t timeout</code> | Specifies the timeout period in seconds. When not specified, the default value is 10 seconds. |
| <code>--help</code> | Prints out help text for the <code>dumpsys</code> tool. |
| <code>-l</code> | Outputs a complete list of system services that you can use with <code>dumpsys</code> . |
| <code>--skip services</code> | Specifies the <code>services</code> that you do not want to include in the output. |
| <code>service [arguments]</code> | Specifies the <code>service</code> that you want to output. Some services may allow you to pass optional <code>arguments</code> . You can learn about these optional arguments by passing the <code>-h</code> option with the service, as shown below: <div>adb shell dumpsys procstats -h</div> |
| <code>-c</code> | When specifying certain services, append this option to output data in a machine-friendly format. |
| <code>-h</code> | For certain services, append this option to see help text and additional options for that service. |

Inspect input diagnostics

Specifying the `input` service, as shown below, dumps the state of the system’s input devices, such as keyboards and touchscreens, and the processing of input events.

```
adb shell dumpsys input
```

describe the type of information you typically see.

Event hub state

The following is a sample of what you might see when inspecting the **Event Hub State** of the input diagnostics:

```
INPUT MANAGER (dumpsys input)

Event Hub State:
  BuiltInKeyboardId: -2
  Devices:
    -1: Virtual
      Classes: 0x40000023
      Path:
      Descriptor: a718a782d34bc767f4689c232d64d527998ea7fd
      Location:
      ControllerNumber: 0
      UniqueId:
      Identifier: bus=0x0000, vendor=0x0000, product=0x0000, version=0x0000
      KeyLayoutFile: /system/usr/keylayout/Generic.kl
      KeyCharacterMapFile: /system/usr/keychars/Virtual.kcm
      ConfigurationFile:
      HaveKeyboardLayoutOverlay: false
    1: msm8974-taiko-mtp-snd-card Headset Jack
      Classes: 0x00000080
      Path: /dev/input/event5
      Descriptor: c8e3782483b4837ead6602e20483c46ff801112c
      Location: ALSA
      ControllerNumber: 0
      UniqueId:
      Identifier: bus=0x0000, vendor=0x0000, product=0x0000, version=0x0000
      KeyLayoutFile:
      KeyCharacterMapFile:
      ConfigurationFile:
      HaveKeyboardLayoutOverlay: false
    2: msm8974-taiko-mtp-snd-card Button Jack
      Classes: 0x00000001
      Path: /dev/input/event4
      Descriptor: 96fe62b244c555351ec576b282232e787fb42bab
      Location: ALSA
      ControllerNumber: 0
      UniqueId:
      Identifier: bus=0x0000, vendor=0x0000, product=0x0000, version=0x0000
      KeyLayoutFile: /system/usr/keylayout/msm8974-taiko-mtp-snd-card_Button_Jack.kl
```

This site uses cookies to store your preferences for site-specific language and display.

```

    ConfigurationFile:
    HaveKeyboardLayoutOverlay: false
3: hs_detect
    Classes: 0x00000081
    Path: /dev/input/event3
    Descriptor: 485d69228e24f5e46da1598745890b214130dbc4
    Location:
    ControllerNumber: 0
    UniqueId:
    Identifier: bus=0x0000, vendor=0x0001, product=0x0001, version=0x0001
    KeyLayoutFile: /system/usr/keylayout/hs_detect.kl
    KeyCharacterMapFile: /system/usr/keychars/hs_detect.kcm
    ConfigurationFile:
    HaveKeyboardLayoutOverlay: false
...

```

Input reader state

The `InputReader` is responsible for decoding input events from the kernel. Its state dump shows information about how each input device is configured and recent state changes that have occurred, such as key presses or touches on the touch screen.

The following sample shows the output for a touch screen. Note the information about the resolution of the device and the calibration parameters that were used.

```

Input Reader State
...
Device 6: Melfas MMSxxx Touchscreen
  IsExternal: false
  Sources: 0x00001002
  KeyboardType: 0
  Motion Ranges:
    X: source=0x00001002, min=0.000, max=719.001, flat=0.000, fuzz=0.999
    Y: source=0x00001002, min=0.000, max=1279.001, flat=0.000, fuzz=0.999
    PRESSURE: source=0x00001002, min=0.000, max=1.000, flat=0.000, fuzz=0.000
    SIZE: source=0x00001002, min=0.000, max=1.000, flat=0.000, fuzz=0.000
    TOUCH_MAJOR: source=0x00001002, min=0.000, max=1468.605, flat=0.000, fuzz=0.000
    TOUCH_MINOR: source=0x00001002, min=0.000, max=1468.605, flat=0.000, fuzz=0.000
    TOOL_MAJOR: source=0x00001002, min=0.000, max=1468.605, flat=0.000, fuzz=0.000
    TOOL_MINOR: source=0x00001002, min=0.000, max=1468.605, flat=0.000, fuzz=0.000
  Touch Input Mapper:
    Parameters:
      GestureMode: spots
      DeviceType: touchScreen

```

This site uses cookies to store your preferences for site-specific language and display.

```
OrientationAware: true
Raw Touch Axes:
  X: min=0, max=720, flat=0, fuzz=0, resolution=0
  Y: min=0, max=1280, flat=0, fuzz=0, resolution=0
  Pressure: min=0, max=255, flat=0, fuzz=0, resolution=0
  TouchMajor: min=0, max=30, flat=0, fuzz=0, resolution=0
  TouchMinor: unknown range
  ToolMajor: unknown range
  ToolMinor: unknown range
  Orientation: unknown range
  Distance: unknown range
  TiltX: unknown range
  TiltY: unknown range
  TrackingId: min=0, max=65535, flat=0, fuzz=0, resolution=0
  Slot: min=0, max=9, flat=0, fuzz=0, resolution=0
Calibration:
  touch.size.calibration: diameter
  touch.size.scale: 10.000
  touch.size.bias: 0.000
  touch.size.isSummed: false
  touch.pressure.calibration: amplitude
  touch.pressure.scale: 0.005
  touch.orientation.calibration: none
  touch.distance.calibration: none
SurfaceWidth: 720px
SurfaceHeight: 1280px
SurfaceOrientation: 0
Translation and Scaling Factors:
  XScale: 0.999
  YScale: 0.999
  XPrecision: 1.001
  YPrecision: 1.001
  GeometricScale: 0.999
  PressureScale: 0.005
  SizeScale: 0.033
  OrientationCenter: 0.000
  OrientationScale: 0.000
  DistanceScale: 0.000
  HaveTilt: false
  TiltXCenter: 0.000
  TiltXScale: 0.000
  TiltYCenter: 0.000
  TiltYScale: 0.000
Last Button State: 0x00000000
Last Raw Touch: pointerCount=0
Last Cooked Touch: pointerCount=0
```

This site uses cookies to store your preferences for site-specific language and display.

such as the tap interval.

Configuration:

```
ExcludedDeviceNames: []
VirtualKeyQuietTime: 0.0ms
PointerVelocityControlParameters: scale=1.000, lowThreshold=500.000, highThreshold=
WheelVelocityControlParameters: scale=1.000, lowThreshold=15.000, highThreshold=50.
PointerGesture:
  Enabled: true
  QuietInterval: 100.0ms
  DragMinSwitchSpeed: 50.0px/s
  TapInterval: 150.0ms
  TapDragInterval: 300.0ms
  TapSlop: 20.0px
  MultitouchSettleInterval: 100.0ms
  MultitouchMinDistance: 15.0px
  SwipeTransitionAngleCosine: 0.3
  SwipeMaxWidthRatio: 0.2
  MovementSpeedRatio: 0.8
  ZoomSpeedRatio: 0.3
```

Input dispatcher state

The `InputDispatcher` is responsible for sending input events to applications. As shown in the sample output below, its state dump shows information about which window is being touched, the state of the input queue, whether an ANR is in progress, and so on.

Input Dispatcher State:

```
DispatchEnabled: 1
DispatchFrozen: 0
FocusedApplication: <null>
FocusedWindow: name='Window{3fb06dc3 u0 StatusBar}'
TouchStates: <no displays touched>
Windows:
  0: name='Window{357bbbfe u0 SearchPanel}', displayId=0, paused=false, hasFocus=fa
  1: name='Window{3b14c0ca u0 NavigationBar}', displayId=0, paused=false, hasFocus=
  2: name='Window{2c7e849c u0 com.vito.lux}', displayId=0, paused=false, hasFocus=f
  ...
MonitoringChannels:
  0: 'WindowManager (server)'
RecentQueue: length=10
  MotionEvent(deviceId=4, source=0x00001002, action=2, flags=0x00000000, metaState=
  MotionEvent(deviceId=4, source=0x00001002, action=1, flags=0x00000000, metaState=
```

This site uses cookies to store your preferences for site-specific language and display.

```

...
PendingEvent: <none>
InboundQueue: <empty>
ReplacedKeys: <empty>
Connections:
  0: channelName='WindowManager (server)', windowName='monitor', status=NORMAL, mor
    OutboundQueue: <empty>
    WaitQueue: <empty>
  1: channelName='278c1d65 KeyguardScrim (server)', windowName='Window{278c1d65 u0
    OutboundQueue: <empty>
    WaitQueue: <empty>
  2: channelName='357bbbfe SearchPanel (server)', windowName='Window{357bbbfe u0 Se
    OutboundQueue: <empty>
    WaitQueue: <empty>
  ...
AppSwitch: not pending
  7: channelName='2280455f com.google.android.gm/com.google.android.gm.Conversation
    OutboundQueue: <empty>
    WaitQueue: <empty>
  8: channelName='1a7be08a com.android.systemui/com.android.systemui.recents.Recent
    OutboundQueue: <empty>
    WaitQueue: <empty>
  9: channelName='3b14c0ca NavigationBar (server)', windowName='Window{3b14c0ca u0
    OutboundQueue: <empty>
    WaitQueue: <empty>
  ...
Configuration:
  KeyRepeatDelay: 50.0ms
  KeyRepeatTimeout: 500.0ms

```

Things to check for

The following is a list of things to consider when inspecting the various output for the `input` service:

Event hub state:

- All of the input devices you expect are present.
- Each input device has an appropriate key layout file, key character map file, and input device configuration file. If the files are missing or contain syntax errors, then they will not be loaded.
- Each input device is classified correctly. The bits in the `Classes` field correspond to flags in `EventHub.h`, such as `INPUT_DEVICE_CLASS_TOUCH_MT`.
- The `BuiltInKeyboardId` is correct. If the device does not have a built-in keyboard, then the id must be

This site uses cookies to store your preferences for site-specific language and display.

- If you observe that the `BuiltInKeyboardId` is not `-2` but it should be, then you are missing a key character map file for a special function keypad somewhere. Special function keypad devices should have key character map files that contain just the line `type SPECIAL_FUNCTION` (that's what in the `tuna-gpio-keykad.kcm` file we see mentioned above).

Input reader state:

- All of the expected input devices are present.
- Each input device is configured correctly. In particular, check that the touch screen and joystick axes are correct.

Input dispatcher state:

- All input events are processed as expected.
- After touching the touch screen and running `dumpsys` at the same time, the `TouchStates` line correctly identifies the window that you are touching.

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:

```
adb shell dumpsys gfxinfo package-name
```

You can also include the `framestats` option to provide even more detailed frame timing information from recent frames, so that you can track down and debug problems more accurately, shown below:

```
adb shell dumpsys gfxinfo package-name framestats
```

To learn more about using `gfxinfo` and `framestats` to integrate UI performance measurements into your testing practices, go to [Testing UI Performance \(https://developer.android.com/training/testing/performance.html\)](https://developer.android.com/training/testing/performance.html).

This site uses cookies to store your preferences for site-specific language and display.

Specifying the `netstats` service provides network usage statistics collected since the previous device booted up. To output additional information, such as detailed unique user ID (UID) information, include the `detail` option, as follows:

```
adb shell dumpsys netstats detail
```

The output varies depending on the version of Android running on the connected device. The sections below describe the type of information you typically see.

Active interfaces and active UID interfaces

The following sample output lists the active interfaces and active UID interfaces of the connected device. In most cases, the information for active interfaces and active UID interfaces is the same.

```
Active interfaces:
  iface=wlan0 ident=[{type=WIFI, subType=COMBINED, networkId="Guest"}]
Active UID interfaces:
  iface=wlan0 ident=[{type=WIFI, subType=COMBINED, networkId="Guest"}]
```

'Dev' and 'Xt' statistics

The following is a sample output for the Dev statistics section:

```
Dev stats:
  Pending bytes: 1798112
  History since boot:
  ident=[{type=WIFI, subType=COMBINED, networkId="Guest", metered=false}] uid=-1 set=
  NetworkStatsHistory: bucketDuration=3600
    st=1497891600 rb=1220280 rp=1573 tb=309870 tp=1271 op=0
    st=1497895200 rb=29733 rp=145 tb=85354 tp=185 op=0
    st=1497898800 rb=46784 rp=162 tb=42531 tp=192 op=0
    st=1497902400 rb=27570 rp=111 tb=35990 tp=121 op=0
Xt stats:
  Pending bytes: 1771782
  History since boot:
  ident=[{type=WIFI, subType=COMBINED, networkId="Guest", metered=false}] uid=-1 set=
  NetworkStatsHistory: bucketDuration=3600
    st=1497891600 rb=1219598 rp=1557 tb=291628 tp=1255 op=0
    st=1497895200 rb=29623 rp=142 tb=82699 tp=182 op=0
    st=1497898800 rb=46684 rp=160 tb=39756 tp=191 op=0
    st=1497902400 rb=27528 rp=110 tb=34366 tp=120 op=0
```

This site uses cookies to store your preferences for site-specific language and display.

UID stats

The following is a sample of detailed statistics of each UID.

```
UID stats:
  Pending bytes: 744
  Complete history:
  ident=[[type=MOBILE_SUPL, subType=COMBINED, subscriberId=311111...], [type=MOBILE,
    NetworkStatsHistory: bucketDuration=7200000
      bucketStart=1406167200000 activeTime=7200000 rxBytes=4666 rxPackets=7 txBytes=1
  ident=[[type=WIFI, subType=COMBINED, networkId="MySSID"]] uid=10007 set=DEFAULT ta
    NetworkStatsHistory: bucketDuration=7200000
      bucketStart=1406138400000 activeTime=7200000 rxBytes=17086802 rxPackets=15387 t
      bucketStart=1406145600000 activeTime=7200000 rxBytes=2396424 rxPackets=2946 txB
      bucketStart=1406152800000 activeTime=7200000 rxBytes=200907 rxPackets=606 txByt
      bucketStart=1406160000000 activeTime=7200000 rxBytes=826017 rxPackets=1126 txBy
```

To find the UID for your app, run this command: `adb shell dumpsys package your-package-name`. 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.example.myapp | grep userId
```

the 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. 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.

This site uses cookies to store your preferences for site-specific language and display.

interval.

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

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](https://developer.android.com/training/monitoring-device-state/doze-standby.html#testing_doze_and_app_standby) (https://developer.android.com/training/monitoring-device-state/doze-standby.html#testing_doze_and_app_standby).

The command for `batterystats` is as follows:

```
adb shell dumpsys batterystats options
```

To see a list of additional options available to `batterystats`, include the `-h` option. The example below outputs battery usage statistics for a specified app package since the device was last charged:

```
adb shell dumpsys batterystats --charged package-name
```

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

To learn more about using `batterystats` and generating an HTML visualization of the output, which makes it easier to understand and diagnose battery-related issues, read [Profile Battery Usage with Batterystats and Battery Historian](https://developer.android.com/studio/profile/battery-historian.html) (<https://developer.android.com/studio/profile/battery-historian.html>).

This site uses cookies to store your preferences for site-specific language and display.

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
...
9,0,i,dsd,1820451,97,s-,p- 9,0,i,dsd,3517481,98,s-,p-
9,0,l,bt,0,8548446,1000983,8566645,1019182,1418672206045,8541652,994188
9,0,l,gn,0,0,666932,495312,0,0,2104,1444
9,0,l,m,6794,0,8548446,8548446,0,0,0,666932,495312,0,697728,0,0,0,5797,0,0
...
```

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.

The table below describes the various section identifiers you may see:

| | | |
|--|--|--|
| | | |
|--|--|--|

This site uses cookies to store your preferences for site-specific language and display. [OK](#)

| Identifier | | |
|------------|-------------------|---|
| vers | Version | checkin version, parcel version, start platform version, end platform version |
| uid | UID | uid, package name |
| apk | APK | wakeups, APK, service, start time, starts, launches |
| pr | Process | process, user, system, foreground, starts |
| sr | Sensor | sensor number, time, count |
| vib | Vibrator | time, count |
| fg | Foreground | time, count |
| st | State Time | foreground, active, running |
| wl | Wake lock | wake lock, full time, 'f', full count, partial time, 'p', partial count, window time, 'w', window count |
| sy | Sync | sync, time, count |
| jb | Job | job, time, count |
| kwl | Kernel Wake Lock | kernel wake lock, time, count |
| wr | Wakeup Reason | wakeup reason, time, count |
| nt | Network | mobile bytes RX, mobile bytes TX, Wi-Fi bytes RX, Wi-Fi bytes TX, mobile packets RX, mobile packets TX, Wi-Fi packets RX, Wi-Fi packets TX, mobile active time, mobile active count |
| ua | User Activity | other, button, touch |
| bt | Battery | start count, battery realtime, battery uptime, total realtime, total uptime, start clock time, battery screen off realtime, battery screen off uptime |
| dc | Battery Discharge | low, high, screen on, screen off |
| lv | Battery Level | start level, current level |
| wfl | Wi-Fi | full Wi-Fi lock on time, Wi-Fi scan time, Wi-Fi running time, Wi-Fi scan |

This site uses cookies to store your preferences for site-specific language and display.

| | | |
|------|-----------------------------|--|
| gwfl | Global Wi-Fi | Wi-Fi on time, Wi-Fi running time, Wi-Fi idle time, Wi-Fi receive time, Wi-Fi transmit time, Wi-Fi power (mAh) |
| gble | Global Bluetooth | BT idle time, BT receive time, BT transmit time, BT power (mAh) |
| m | Misc | screen on time, phone on time, full wakelock time total, partial wakelock time total, mobile radio active time, mobile radio active adjusted time, interactive time, power save mode enabled time, connectivity changes, device idle mode enabled time, device idle mode enabled count, device idling time, device idling count, mobile radio active count, mobile radio active unknown time |
| gn | Global Network | mobile RX total bytes, mobile TX total bytes, Wi-Fi RX total bytes, Wi-Fi TX total bytes, mobile RX total packets, mobile TX total packets, Wi-Fi RX total packets, Wi-Fi TX total packets |
| br | Screen Brightness | dark, dim, medium, light, bright |
| sst | Signal Scanning Time | signal scanning time |
| sgt | Signal Strength Time | none, poor, moderate, good, great |
| sgc | Signal Strength Count | none, poor, moderate, good, great |
| dct | Data Connection Time | none, GPRS, EDGE, UMTS, CDMA, EVDO_0, EVDO_A, 1xRTT, HSDPA, HSUPA, HSPA, IDEN, EVDO_B, LTE, EHRPD, HSPAP, other |
| dcc | Data Connection Count | none, GPRS, EDGE, UMTS, CDMA, EVDO_0, EVDO_A, 1xRTT, HSDPA, HSUPA, HSPA, IDEN, EVDO_B, LTE, EHRPD, HSPAP, other |
| wst | Wi-Fi State Time | off, off scanning, on no networks, on disconnected, on connected STA, on connected P2P, on connected STA P2P, soft AP |
| wsc | Wi-Fi State Count | off, off scanning, on no networks, on disconnected, on connected STA, on connected P2P, on connected STA P2P, soft AP |
| wsst | Wi-Fi Supplicant State Time | invalid, disconnected, interface disabled, inactive, scanning, authenticating, associating, associated, four-way handshake, group handshake, completed, dormant, uninitialized |

This site uses cookies to store your preferences for site-specific language and display.

| | | |
|------|------------------------------|--|
| wssc | Wi-Fi Supplicant State Count | invalid, disconnected, interface disabled, inactive, scanning, authenticating, associating, associated, four-way handshake, group handshake, completed, dormant, uninitialized |
| wsgt | Wi-Fi Signal Strength Time | none, poor, moderate, good, great |
| wsgc | Wi-Fi Signal Strength Count | none, poor, moderate, good, great |
| bst | Bluetooth State Time | inactive, low, med, high |
| bsc | Bluetooth State Count | inactive, low, med, high |
| pws | Power Use Summary | battery capacity, computed power, minimum drained power, maximum drained power |
| pwi | Power Use Item | label, mAh |
| dsd | Discharge Step | duration, level, screen, power-save |
| csd | Charge Step | duration, level, screen, power-save |
| dtr | Discharge Time Remaining | battery time remaining |
| ctr | Charge Time Remaining | charge time remaining |

Note: Prior to Android 6.0, power use for Bluetooth radio, cellular radio, and Wi-Fi was tracked in the *m* (Misc) section category. In Android 6.0 and higher, power use for these components is tracked in the *pwi* (Power Use Item) section with individual labels (*wifi*, *blue*, *cell*) for each component.

View memory allocations

You can inspect your app's memory usage in one of two ways: over a period of time using `procstats` or at a particular snapshot in time using `meminfo`. The sections below show you how to use either method.

This site uses cookies to store your preferences for site-specific language and display

OK

procstats

procstats makes it possible to see how your app is behaving over time—including how long it runs in the background and how much memory it uses during that time. It helps you quickly find inefficiencies and misbehaviors in your app, such as memory leaks, that can affect how it performs, especially when running on low-memory devices. Its state dump displays statistics about every application's runtime, proportional set size (PSS) and unique set size (USS).

To get application memory usage stats over the last three hours, in human-readable format, run the following command:

```
adb shell dumpsys procstats --hours 3
```

As can be seen in the example below, the output displays what percentage of time the application was running, and the PSS and USS as **minPSS-avgPSS-maxPSS/minUSS-avgUSS-maxUSS** over the number of samples.

```
AGGREGATED OVER LAST 3 HOURS:
* com.android.systemui / u0a20 / v22:
    TOTAL: 100% (109MB-126MB-159MB/108MB-125MB-157MB over 18)
    Persistent: 100% (109MB-126MB-159MB/108MB-125MB-157MB over 18)
* com.android.nfc / 1027 / v22:
    TOTAL: 100% (17MB-17MB-17MB/16MB-16MB-16MB over 18)
    Persistent: 100% (17MB-17MB-17MB/16MB-16MB-16MB over 18)
* android.process.acore / u0a4 / v22:
    TOTAL: 100% (14MB-15MB-15MB/14MB-14MB-14MB over 20)
    Imp Fg: 100% (14MB-15MB-15MB/14MB-14MB-14MB over 20)
...
* com.coulombtech / u0a106 / v26:
    TOTAL: 0.01%
    Receiver: 0.01%
    (Cached): 21% (4.9MB-5.0MB-5.2MB/3.8MB-3.9MB-4.1MB over 2)
* com.softcoil.mms / u0a86 / v32:
    TOTAL: 0.01%
    (Cached): 0.25%
* com.udemy.android / u0a91 / v38:
    TOTAL: 0.01%
    Receiver: 0.01%
```

This site uses cookies to store your preferences for site-specific language and display.


```
...
Run time Stats:
  SOff/Norm: +32m52s226ms
  SOn /Norm: +2h10m8s364ms
    Mod : +17s930ms
    TOTAL: +2h43m18s520ms

Memory usage:
  Kernel : 265MB (38 samples)
  Native : 73MB (38 samples)
  Persist: 262MB (90 samples)
  Top    : 190MB (325 samples)
  ImpFg  : 204MB (569 samples)
  ImpBg  : 754KB (345 samples)
  Service: 93MB (1912 samples)
  Receivr: 227KB (1169 samples)
  Home   : 66MB (12 samples)
  LastAct: 30MB (255 samples)
  CchAct : 220MB (450 samples)
  CchCAct: 193MB (71 samples)
  CchEmty: 182MB (652 samples)
  Cached : 58MB (38 samples)
  Free   : 60MB (38 samples)
  TOTAL  : 1.9GB
  ServRst: 50KB (278 samples)

      Start time: 2015-04-08 13:44:18
Total elapsed time: +2h43m18s521ms (partial) libart.so
```

meminfo

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

```
adb shell dumpsys meminfo package_name|pid [-d]
```

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

The output lists all of your app's current allocations, measured in kilobytes.

When inspecting this information, you should be familiar with the following types of allocation:

Private (Clean and Dirty) RAM

This site uses cookies to store your preferences for site-specific language and display.

can reclaim when your app's process is destroyed. Generally, the most important portion of this is *private dirty* RAM, which is the most expensive because it is used by only your process and its contents exist only in RAM so can't be paged to storage (because Android does not use swap). All Dalvik and native heap allocations you make will be private dirty RAM; Dalvik and native allocations you share with the Zygote process are shared dirty RAM.

Proportional Set Size (PSS)

This is a measurement of your app's RAM use that takes into account sharing pages across processes. Any RAM pages that are unique to your process directly contribute to its PSS value, while pages that are shared with other processes contribute to the PSS value only in proportion to the amount of sharing. For example, a page that is shared between two processes will contribute half of its size to the PSS of each process.

A nice characteristic of the PSS measurement is that you can add up the PSS across all processes to determine the actual memory being used by all processes. This means PSS is a good measure for the actual RAM weight of a process and for comparison against the RAM use of other processes and the total available RAM.

For example, below is the output for Map's process on a Nexus 5 device. There is a lot of information here, but key points for discussion are listed below.

```
adb shell dumpsys meminfo com.google.android.apps.maps -d
```

Note: The information you see might vary slightly from what is shown here, because some details of the output differ across platform versions.

** MEMINFO in pid 18227 [com.google.android.apps.maps] **

| | Pss Total | Private Dirty | Private Clean | Swapped Dirty | Heap Size | Heap Alloc | Heap Free |
|--------------|--------------|------------------|------------------|------------------|--------------|---------------|--------------|
| | ----- | ----- | ----- | ----- | ----- | ----- | ----- |
| Native Heap | 10468 | 10408 | 0 | 0 | 20480 | 14462 | 6017 |
| Dalvik Heap | 34340 | 33816 | 0 | 0 | 62436 | 53883 | 8553 |
| Dalvik Other | 972 | 972 | 0 | 0 | | | |
| Stack | 1144 | 1144 | 0 | 0 | | | |
| Gfx dev | 35300 | 35300 | 0 | 0 | | | |
| Other dev | 5 | 0 | 4 | 0 | | | |
| .so mmap | 1943 | 504 | 188 | 0 | | | |
| .apk mmap | 598 | 0 | 136 | 0 | | | |
| .ttf mmap | 134 | 0 | 68 | 0 | | | |
| .dex mman | 3908 | 0 | 3904 | 0 | | | |

```

.art mmap      2037      1784      28        0
Other mmap      30        4        0        0
EGL mtrack    73072     73072      0        0
GL mtrack     51044     51044      0        0
Unknown       185       184        0        0
TOTAL        216524    208232    4384      0      82916     68345     14570

```

Dalvik Details

```

.Heap      6568      6568      0        0
.LOS      24771     24404      0        0
.GC        500       500        0        0
.JITCache   428       428        0        0
.Zygote    1093       936        0        0
.NonMoving  1908      1908        0        0
.IndirectRef  44        44         0        0

```

Objects

```

Views:      90      ViewRootImpl:    1
AppContexts: 4      Activities:      1
Assets:      2      AssetManagers:   2
Local Binders: 21    Proxy Binders:   28
Parcel memory: 18    Parcel count:    74
Death Recipients: 2  OpenSSL Sockets: 2

```

Here is an older dumpsys on Dalvik of the gmail app:

**** MEMINFO in pid 9953 [com.google.android.gm] ****

| | Pss Total | Pss Clean | Shared Dirty | Private Dirty | Shared Clean | Private Clean | Heap Size | Heap Alloc | Heap Free |
|--------------|--------------|--------------|-----------------|------------------|-----------------|------------------|--------------|---------------|--------------|
| Native Heap | 0 | 0 | 0 | 0 | 0 | 0 | 7800 | 7637(6) | 126 |
| Dalvik Heap | 5110(3) | 0 | 4136 | 4988(3) | 0 | 0 | 9168 | 8958(6) | 210 |
| Dalvik Other | 2850 | 0 | 2684 | 2772 | 0 | 0 | | | |
| Stack | 36 | 0 | 8 | 36 | 0 | 0 | | | |
| Cursor | 136 | 0 | 0 | 136 | 0 | 0 | | | |
| Ashmem | 12 | 0 | 28 | 0 | 0 | 0 | | | |
| Other dev | 380 | 0 | 24 | 376 | 0 | 4 | | | |
| .so mmap | 5443(5) | 1996 | 2584 | 2664(5) | 5788 | 1996(5) | | | |
| .apk mmap | 235 | 32 | 0 | 0 | 1252 | 32 | | | |
| .ttf mmap | 36 | 12 | 0 | 0 | 88 | 12 | | | |
| .dex mmap | 3019(5) | 2148 | 0 | 0 | 8936 | 2148(5) | | | |
| Other mmap | 107 | 0 | 8 | 8 | 324 | 68 | | | |
| Unknown | 6994(4) | 0 | 252 | 6992(4) | 0 | 0 | | | |
| TOTAL | 24358(1) | 4188 | 9724 | 17972(2) | 16388 | 4260(2) | 16968 | 16595 | 336 |

Objects

This site uses cookies to store your preferences for site-specific language and display.

```

    AppContexts:      6(7)      Activities:      2(7)
      Assets:         2        AssetManagers:    2
    Local Binders:    64        Proxy Binders:   34
  Death Recipients:   0
    OpenSSL Sockets:   1

SQL
    MEMORY_USED:      1739
  PAGECACHE_OVERFLOW: 1164      MALLOC_SIZE:      62

```

In general, be concerned with only the **Pss Total** and **Private Dirty** columns. In some cases, the **Private Clean** and **Heap Alloc** columns also offer interesting data. More information about the different memory allocations (the rows) you should observe follows:

Dalvik Heap

The RAM used by Dalvik allocations in your app. The **Pss Total** includes all Zygote allocations (weighted by their sharing across processes, as described in the PSS definition above). The **Private Dirty** number is the actual RAM committed to only your app's heap, composed of your own allocations and any Zygote allocation pages that have been modified since forking your app's process from Zygote.

Note: On newer platform versions that have the **Dalvik Other** section, the **Pss Total** and **Private Dirty** numbers for Dalvik Heap do not include Dalvik overhead such as the just-in-time compilation (JIT) and GC bookkeeping, whereas older versions list it all combined under **Dalvik**.

The **Heap Alloc** is the amount of memory that the Dalvik and native heap allocators keep track of for your app. This value is larger than **Pss Total** and **Private Dirty** because your process was forked from Zygote and it includes allocations that your process shares with all the others.

.so mmap and .dex mmap

The RAM being used for mapped **.so** (native) and **.dex** (Dalvik or ART) code. The **Pss Total** number includes platform code shared across apps; the **Private Clean** is your app's own code. Generally, the actual mapped size will be much larger—the RAM here is only what currently needs to be in RAM for code that has been executed by the app. However, the **.so mmap** has a large private dirty, which is due to fix-ups to the native code when it was loaded into its final address.

.oat mmap

This site uses cookies to store your preferences for site-specific language and display.

are commonly used by multiple apps. This image is shared across all apps and is unaffected by particular apps.

.art mmap

This is the amount of RAM used by the heap image which is based off of the preloaded classes which are commonly used by multiple apps. This image is shared across all apps and is unaffected by particular apps. Even though the ART image contains **Object** (<https://developer.android.com/reference/java/lang/Object.html>) instances, it does not count towards your heap size.

.Heap (only with -d flag)

This is the amount of heap memory for your app. This excludes objects in the image and large object spaces, but includes the zygote space and non-moving space.

.LOS (only with -d flag)

This is the amount of RAM used by the ART large object space. This includes zygote large objects. Large objects are all primitive array allocations larger than 12KB.

.GC (only with -d flag)

This is the amount of internal GC accounting overhead for your app. There is not really any way to reduce this overhead.

.JITCache (only with -d flag)

This is the amount of memory used by the JIT data and code caches. Typically, this is zero since all of the apps will be compiled at installed time.

.Zygote (only with -d flag)

This is the amount of memory used by the zygote space. The zygote space is created during device startup and is never allocated into.

.NonMoving (only with -d flag)

This is the amount of RAM used by the ART non-moving space. The non-moving space contains special non-movable objects such as fields and methods. You can reduce this section by using fewer

This site uses cookies to store your preferences for site-specific language and display

.IndirectRef (only with -d flag)

This is the amount of RAM used by the ART indirect reference tables. Usually this amount is small, but if it is too high, it might be possible to reduce it by reducing the number of local and global JNI references used.

Unknown

Any RAM pages that the system could not classify into one of the other more specific items. Currently, this contains mostly native allocations, which cannot be identified by the tool when collecting this data due to Address Space Layout Randomization (ASLR). Like the Dalvik heap, the **Pss Total** for Unknown takes into account sharing with Zygote, and **Private Dirty** is unknown RAM dedicated to only your app.

TOTAL

The total Proportional Set Size (PSS) RAM used by your process. This is the sum of all PSS fields above it. It indicates the overall memory weight of your process, which can be directly compared with other processes and the total available RAM.

The **Private Dirty** and **Private Clean** are the total allocations within your process, which are not shared with other processes. Together (especially **Private Dirty**), this is the amount of RAM that will be released back to the system when your process is destroyed. Dirty RAM is pages that have been modified and so must stay committed to RAM (because there is no swap); clean RAM is pages that have been mapped from a persistent file (such as code being executed) and so can be paged out if not used for a while.

ViewRootImpl

The number of root views that are active in your process. Each root view is associated with a window, so this can help you identify memory leaks involving dialogs or other windows.

AppContexts and Activities

The number of app **Context** (<https://developer.android.com/reference/android/content/Context.html>) and **Activity** (<https://developer.android.com/reference/android/app/Activity.html>) objects that currently live in your process. This can help you to quickly identify leaked **Activity** (<https://developer.android.com/reference/android/app/Activity.html>) objects that can't be garbage

This site uses cookies to store your preferences for site-specific language and displ.

Note: A **View** (<https://developer.android.com/reference/android/view/View.html>) or **Drawable** (<https://developer.android.com/reference/android/graphics/drawable/Drawable.html>) object also holds a reference to the **Activity** (<https://developer.android.com/reference/android/app/Activity.html>) that it's from, so holding a **View** (<https://developer.android.com/reference/android/view/View.html>) or **Drawable** (<https://developer.android.com/reference/android/graphics/drawable/Drawable.html>) object can also lead to your app leaking an **Activity** (<https://developer.android.com/reference/android/app/Activity.html>).