

# Browsing and debugging AOSP code using Android Studio

Chris Simmonds, 2net

# Overview

- Importing AOSP into Android Studio
- Debugging

# Creating IDEGen

- AOSP has a tool named `IDEGen` that will generate project files suitable for IntelliJ IDEA (including Android Studio)

Details: `$AOSP/development/tools/idegen/README`

Set up the AOSP environment in the normal way:

```
$ cd aosp
$ . build/envsetup.sh
$ lunch
```

Build IDEGen:

```
$ m idegen
```

Creates `out/host/linux-x86/framework/idegen.jar`

# Creating project files

Then, use `idegen.jar` to generate project files (ignore the "Permission denied" messages):

```
$ development/tools/idegen/idegen.sh
find: 'out/target/product/marvin/recovery/root/d': Permission denied
find: 'out/target/product/marvin/root/d': Permission denied
Read excludes: 23ms
Traversed tree: 148105ms
```

Creates an IntelliJ Project File: `android.ipr`  
and an IntelliJ Module List: `android.iml`

**Note:** you have to rerun this step every time you add directories inside the `$AOSP` directory

# Configuring Android Studio

Indexing the whole of AOSP takes a lot of resources

## Increase heap to 5 GB

In Help->Edit Custom VM Options, add:

```
-Xmx5g
```

For reference, the VM Options are stored in

```
$HOME/.config/Google/AndroidStudio2021.2/studio64.vmoptions
```

for the current version of Studio, 2021.2 "Chipmunk"

# Importing AOSP code

In Android Studio, open `$AOSP/android.ipr`

Wait for it to complete "Updating indexes" (background task, bottom right); takes a long time

Now you can use Studio to browse the code, search for classes, etc.

Some people consider this a better alternative to using vim

# Debugging - JDWP

- Android uses JDWP (Java Debug Wire Protocol) for debugging
- JDWP is enabled by default for all components in userdebug and eng builds
- In user builds, enable JDWP per application by adding `android:debuggable="true"` to the `AndroidManifest.xml`

```
$ ps -ATp 3694
USER      PID    TID   PPID    VSZ    RSS WCHAN   ADDR S  CMD
u0_a65    3694   3694   355 1409680 203024 0         0 S  droid.deskclock
u0_a65    3694   3697   355 1409680 203024 0         0 S  Signal Catcher
u0_a65    3694   3698   355 1409680 203024 0         0 S  perfetto_hprof_
u0_a65    3694   3699   355 1409680 203024 0         0 S  ADB-JDWP Connec <----
u0_a65    3694   3700   355 1409680 203024 0         0 S  Jit thread pool
[...]
```

# system\_server

- system\_server is a Java program (not an app) that implements the core functions of the framework
- It also has a JDWP thread in userdebug and eng builds

```
$ ps -ATp 765
USER      PID    TID   PPID     VSZ    RSS  WCHAN      ADDR S  CMD
system    765    765   355 2427732 327936 0          0 S  system_server
system    765    782   355 2427732 327936 0          0 S  Signal Catcher
system    765    783   355 2427732 327936 0          0 S  perfetto_hprof_
system    765    784   355 2427732 327936 0          0 S  ADB-JDWP Connec <-----
[...]
```



# adb jdwp

- You can get a list of PIDs which have JDWP threads:

```
$ adb jdwp
765          <----- system_server
1850
[...]
3694        <----- deskclock
```

Unfortunately, only lists the PIDs. You have to use `ps -A` to find the command

# DDMS

- DDMS is the Dalvik Debug Monitor Service, an old debug tool
- Useful for debugging: lists processes with a JDWP thread, and forwards the port as localhost:8700

The screenshot displays the Dalvik Debug Monitor (DDMS) application. The main window is titled "Dalvik Debug Monitor" and has a menu bar with "File", "Edit", "Actions", and "Device". Below the menu bar is a toolbar with icons for connection, refresh, and other actions. The main area is divided into two panes. The left pane shows a tree view of processes under the heading "Name". The right pane shows details for the selected process, including "Info", "Threads", "VM Heap", and "Allocation Tracker". The bottom pane shows a message log with a search bar and a table of messages.

Name	PID	TID	Application	Tag
google-paranoid_android-0.0.0.0:6520	Online		12, debug	
com.android.google.gce.gceservice	3041		8600	
com.android.providers.media.module	2437		8601	
com.android.launcher3	2086		8602	
android.ext.services	2378		8603	
com.android.se	1994		8604	
com.android.networkstack.process	1963		8605	
com.android.systemui	1868		8606	
com.android.statementservice	3309		8607	
com.android.deskclock	3694		8608	
com.android.phone	2036		8609	
com.android.carrierconfig	2968		8610	
com.android.bluetooth	1850		8611	
system_process	765		8612 / 8700	
com.android.inputmethod.latin	2334		8613	

Info

DDM-aware? yes  
App description: system\_process  
VM version: Dalvik v2.1.0  
Process ID: 765  
Supports Profiling Control: Yes  
Supports HPROF Control: Yes  
ABI Flavor: 32-bit (x86)  
JVM Flags: CheckJNI=false

Search for messages. Accepts java regexes. Prefix with pid., app., tag: or text: to filter

verbose

Le	Time	PID	TID	Application	Tag	Text
						estamp=5296567 mSlee 0, 0] mRxTimeMs=0}

# Debuggers for Android

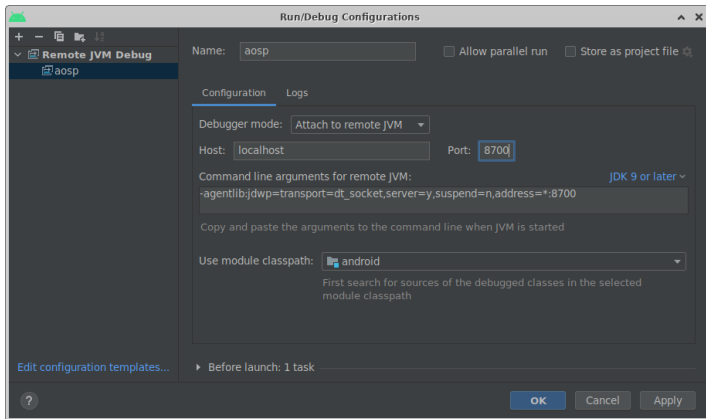
- Any debugger that is compatible with JDWP
- For example
  - Android Studio (IntelliJ IDEA)
  - JDB
- Just need to attach to the JDWP port (8700 if using DDMS)

# Debugging using Android Studio

In Android Studio, create a debug configuration (Run -> Edit Configurations...)

Add a new **Remote JVM Debug**, for example called 'aosp'

Set Port to **8700**



# Debugging using Android Studio

Boot target device

Run DDMS and select process: for system\_server, select **system\_process**

In Android Studio, Run -> Debug 'aosp'

Check that you see this in the debug window

```
Connected to the target VM, address: 'localhost:8700', transport: 'socket'
```

# Debugging using JDB

- JDB is a command-line debugger, part of the JDK
- Pros: easy to set up, nothing new to install, no need for IDEGen, fast
- Cons: a lot of typing

More details to follow, probably at the next meetup