Android
Open Source Project

Stefan Mitev

Stefan Mitev

- Windsurfer
- Former web & desktop developer
- Android developer
- Working at Pixplicity

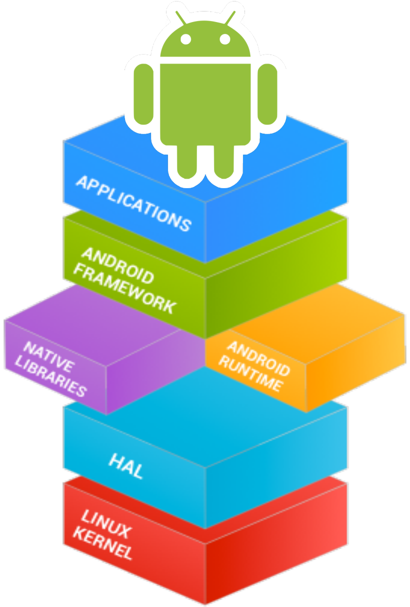
Overview

- The Android stack
- Application fundamentals
- Application manifest

- AOSP - what, who, why, how
- IDE integration

Android

Android Stack



launcher, browser, gallery, calculator

content providers, managers
(such as Activity-, Location-, PackageManager)

native & core libs, heart of Android, Dalvik VM

interface between the framework and the hardware drivers
(sensors, graphics, bluetooth, etc.)

the bridge between the software and the hardware
+ wake locks, Binder IPC driver, mobile embedded specific features

Application fundamentals

- Usually apps are written in Java
- Apps are linux users
- Apps live in their own security sandbox
- Each app has its own VM

Application components

- Activities .... screen where UI is drawn
- Services .... for long-running background operations, no UI
- Content Providers .... managing and encapsulating structured data
- Broadcast Receivers .... listeners for system or application events

Application Manifest

- Essential information for the Android System about a particular application
- The PackageManager inspects the intent filters and its list so that the platform know which app is capable of capturing which intents.
- Part of the information is also used by the Google Play Store.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest package="...">

      <uses-permission />
      <uses-feature /> ...

      <application>
            <activity>
                  <intent-filter>
                        <action ... /> <category ... /> <data ... />
                  </intent-filter>
            </activity>

            <service>
                  <intent-filter> ... </intent-filter>
            </service>

            <receiver>
                  <intent-filter> ... </intent-filter>
            </receiver>

            <provider/>
      </application>
</manifest>
```

AOSP

Available for..

SAMSUNG

ASUS®

SONY

htc

Custom distribution? Why?

- modify the Android SDK
- modify existing apps
- add our libraries
- add our system apps
- change boot animation
- customize the user experience
- tailor the platform for specific use case
- etc.

cyanogenmod

amazon.com

fire

PHONE

Establish a build environment

Requirements

- Linux or Mac OS X or Win+VM
- 64bit OS for Android > 2.3.x
- >= 8 GB RAM/Swap
- >= *(guess!)* free space (SSD is a +)
- Python 2.6 - 2.7
- GNU Make 3.81 - 3.82
- JDK 7 for Android >= 5.0
- Git >= 1.7
- Repo tool

Source code organization

The AOSP manifest

```
<manifest>
  <remote name="aosp"
          fetch="https://android.googlesource.com/"/>

  <default revision="refs/tags/android-5.1.1_r1"
          remote="aosp"
          sync-j="4" />

  <project path="frameworks/base" name="
platform/frameworks/base" groups="pdk-cw-fs" />
  <project path="packages/apps/Browser" name="
platform/packages/apps/Browser" />
  <project path="packages/apps/Launcher3" name="
platform/packages/apps/Launcher3" />
</manifest>
```

Prepare Repo

1. Create a bin/ directory in your home directory and include it in your path

```
$ mkdir ~/bin
$ PATH=~/bin:$PATH
```

2. Download the tool

```
$ curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
```

3. Make it executable

```
$ chmod a+x ~/bin/repo
```

Initialize a Repo client

1. Create an empty working directory

```
$ mkdir ~/aosp
```

2. Initialize the Repo client into your working directory, by checking out from a branch/tag*

```
$ cd ~/aosp && repo init -u https://android.googlesource.com/platform/manifest -b android-5.1.1_r1
```

3. Done! Now you should have a .repo subdirectory created.

**List with branches/tags: http://source.android.com/source/build-numbers.html#source-code-tags-and-builds*

Pull the Android Source Tree

1. Execute

```
$ repo sync
```

2. Waaaait for it… :)

Building AOSP

Setup the env

Source from the *envsetup.sh* script

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

Choose a target

Use "*lunch*" to choose what kind of device you want to build for.

```
$ lunch <product_name>[_<sub-product_name>]-<build_variant>
```

ex.

```
$ lunch aosp_grouper-eng
$ lunch aosp_x86_64-eng
```

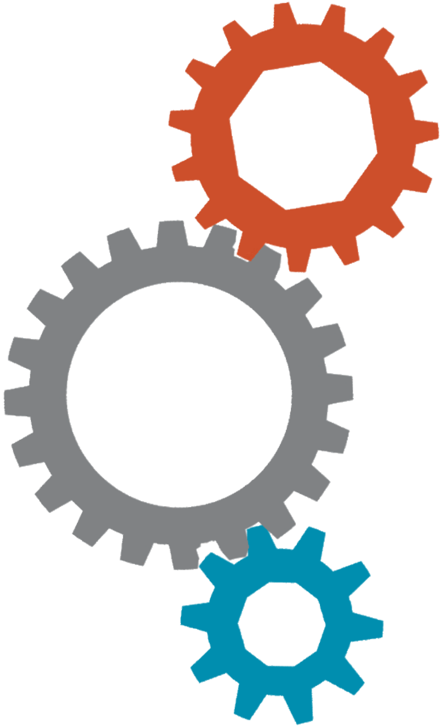
`<product>` - set of modules to be included among various configurations.

- `generic` - default set of packages
- `full` - full set of packages, with all apps and locales
- `aosp` - it actually inherits everything from full
- `sdk` - packages needed to build the SDK

`<build_variant>`

- `user` - variant with limited access that is suited for production
- `userdebug` - like "user" but with root access and debuggability
- `eng` - variant with development configuration with additional debugging tools

Installing drivers



- Drivers for Nexus devices can be downloaded from https://developers.google.com/android/nexus/drivers

What about Nexus 9, though? What's the catch?

- Each set of binaries comes as a self-extracting script in a compressed archive.

In order to make sure the newly installed drivers are properly taken into account after being extracted, we have to exec

```
$ make clean
```

Building the AOSP

```
$ make -j8
```

… waaaaaaait for it…. waait for it….

Flashing a device

Flash a real device

1. Unlock the bootloader*

```
$ fastboot oem unlock
```

2. Boot into fastboot

```
$ adb reboot bootloader
```

3. Flash the images

```
$ fastboot flashall -w
```

Flash an emulator

1. Execute

```
$ emulator
```

* *Depending on the device, it's a matter of executing a simple shell command or using an external software.*

Tips

Use compiler cache for C/C++ code

```
 $ export USE_CCACHE=1
 $ export CACHE_DIR=/<path>/.ccache
 $ ~/aosp/prebuilts/misc/linux-x86/ccache/ccache
-M 100G
```

Build only certain modules

```
 $ cd ~/aosp
 mmm packages/apps/Music
 mmm packages/apps/Music packages/apps/Calendar
```

Only recreate the system image files

```
 $ make snod
```

Syncing the changes directly onto a device

```
 $ adb sync
 $ adb shell stop // Only for framework modules
 $ adb shell start
```

Android Studio Integration

1. Edit *studio.vmoptions* or *studio64.vmoptions* to increase the allocated heap size on startup and its maximum size. *(Use idea[64].vmoptions for IntelliJ)*

   ```
   -Xms750m
   -Xmx800m
   ```

2. Edit *idea.properties* and change the max file size the IDE should provide code assistance for

   ```
   idea.max.intellisense.filesize=5000
   ```

3. Compile the *idegen* tools *(if it's not yet)*

   ```
   $ cd ~/aosp/development/tools/idegen; mm
   ```

4. Create a shadow directory of the working directory

   ```
   $ mkdir ~/aosp-shadow && cd ~/aosp-shadow && lndir ~/aosp
   ```

5. Run the idegen tool

   ```
   $ cd ~/aosp-shadow; development/tools/idegen/idegen.sh
   ```

Android Studio Integration continued

6. Open *android.ipr* with *Android Studio* and you should have the *AOSP* imported.
7. Add *Oracle Java 7 SDK* without any libraries.
8. Navigate to *File->Project structure* and remove all dependencies that end with a *.jar*
9. Go to *Sources* tab and expand *out/target/common/R.*
10. Right click on it and click "*Source*". Then apply the changes.

Note: Consider turning "*Power save mode*" on in order to stop the code inspection.

Thanks!

Stefan Mitev
stefan@pixplicity.com
mr.mitew@gmail.com