INDUSTRIALIZE YOUR ROM COOKING: Good practices
AGENDA

01  What’s this?
02  A good start
03  Create your device
04  Adapt Android
What’s this?
PROFESSIONAL ROM

What’s this?

For professionals

Specific core business features

High quality requirements
PROFESSIONAL ROM

Why Android?

- Open Source with AOSP
- After 4 years: 80% market share
- Sustainable solution
- Multiple targets
PROFESSIONAL ROM

Good practices?

- Android is a very recent OS
- Android/Linux ≠ GNU/Linux

We have seen all the presented examples in production
A GOOD START
How to build?

> source build/envsetup.sh

> lunch

> make -j42
A GOOD START
lunch menu

You're building on Linux

Lunch menu... pick a combo:
1. aosp_arm-eng
2. aosp_x86-eng
3. aosp_mips-eng
4. vbox_x86-eng
5. vbox86p-userdebug
6. vbox86p-eng
7. vbox86t-userdebug
8. vbox86t-eng
9. vbox86tp-userdebug
10. vbox86tp-eng
11. aosp_deb-userdebug
12. aosp_flo-userdebug
13. aosp_grouper-userdebug
14. aosp_tilapia-userdebug
15. mini_armv7a_neon-userdebug
16. mini_mips-userdebug
17. mini_x86-userdebug
18. aosp_hammerhead-userdebug
19. aosp_mako-userdebug
20. aosp_manta-userdebug

Which would you like? [aosp_arm-eng] 9

PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=4.4.2
TARGET_PRODUCT=vbox86tp
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_BUILD_APPS=
TARGET_ARCH=x86
TARGET_ARCH_VARIANT=x86
TARGET_CPU_VARIANT=
HOST_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux:3.11.0-19-generic-x86_64-with-Ubuntu-13.10-saucy
HOST_BUILD_TYPE=release
BUILD_ID=K0T49H
OUT_DIR=out
A GOOD START
Sources organization 1/4

- **bionic/**: Android LibC
- **bootable/**: Bootloader & recovery
- **build/**: The build system
- **cts/**: Compatibility Test Suite: the certification test suite
A GOOD START
Sources organization 2/4

- development/
  - Development tools

- device/
  - Device specific files

- external/
  - External libraries & applications

- frameworks/
  - The Android framework
A GOOD START
Sources organization 3/4

- **libcore/**: Java implementation (Apache Harmony)
- **ndk/**: Native Development Kit
- **out/**: Compilation results
- **packages/**: Android base applications
A GOOD **START**
Sources organization 4/4

- **prebuilt/**
  - Precompiled binaries (toolchain, gdbserver...)

- **sdk/**
  - Software development kit

- **system/**
  - Android system tools: init, toolbox, logcat...

- **vendor/**
  - Vendor directory (libs, apps)
A GOOD START
Version management

REPO !!
A GOOD START
What about the hardware?

A professional device, it’s a whole thing

Work on a ROM depend a lot of the hardware

Depending on the hardware, the same ROM can require a lot more work to be less complete
CREATE YOUR DEVICE
CREATE YOUR DEVICE
Add support for a device

AOSP

device/
vendor/
hardware/
CREATE YOUR **DEVICE**
Manufacturing the target

In `device/$(BRAND)/$(MODEL)`:

- `vendorsetup.sh`
- `anbusu.mk`
- `add_lunch_combo anbusu-eng`
- `BoardConfig.mk`
- `anbusu`
- `device`

**Variables**:
- `BOARD_HAVE_BLUETOOTH`
- `WPA_SUPPLICANT_VERSION`
- `PRODUCT_PACKAGE`
- `PRODUCT_COPY_FILES`
# BoardConfig.mk

PRODUCT_MANUFACTURER := Genymobile
TARGET_ARCH=x86
BUILDROID_X86_NOSSE2 := true
TARGET_CPU_SMP := true
DISABLE_DEXPREOPT := true
TARGET_COMPRESS_MODULE_SYMBOLS := false
TARGET_NO_RECOVERY := true

...
# anbusu.mk

$(call inherit-product,\ device/genymobile/genydemo/device.mk)

$(call inherit-product,\ $(SRC_TARGET_DIR)/product/aosp_base.mk)

PRODUCT_NAME := aosp_anbusu
PRODUCT_DEVICE := anbusu
PRODUCT_BRAND := genybrand
PRODUCT_MODEL := AOSP on Android Builder Summit
PRODUCT_MANUFACTURER := genymanufacturer
# device.mk

PRODUCT_PACKAGES += \
    anbusuapp

PRODUCT_COPY_FILES += \
    device/genymobile/genydemo/init.paug.rc : root/init.anbusu.rc

PRODUCT_PROPERTY_OVERRIDES += \
    ro.genytruc=genyvalue

PRODUCT_DEFAULT_PROPERTY_OVERRIDES +=\n    ro.genydream = genyvalue
CREATE YOUR DEVICE

device/$(BRAND)/$(MODEL)/device.mk

DEVICE_PACKAGE_OVERLAYS += \\
device/genymobile/genydemo/overlay

# the actual meat of the device-specific product definition
$(call inherit-product, \\
device/genymobile/anbusubase/device-common.mk)

# inherit from the non-open-source side, if present
$(call inherit-product-if-exists, \\
vendor/genymobile/genycusto/device-vendor.mk)
CREATE YOUR DEVICE
Customize Android

In device/$(BRAND)/$(MODEL) :

- overlay

- frameworks-ext

- frameworks
Hardware support
Specific to a component
(vs device)

We often have binaries in vendor/ because sources are often proprietaries
ADAPT ANDROID
From bottom to top
Adapt **ANDROID**

First: the kernel
# in init.rc

on boot
    insmod /system/lib/modules/anbusu.ko
    chown system system /dev/anbusu
    chmod 0600 /dev/anbusu

# in ./device/<vendor>/<product>/device.mk

PRODUCT_COPY_FILES := \
device/<vendor>/<product>/anbusu.ko: \
    system/lib/modules/anbusu.ko
adb shell

# ls -l /dev/anbusu
crw------- system system 249, 0 anbusu

# echo Hello > /dev/anbusu

# cat /dev/anbusu
hELLO

#
Adapt ANDROID
Low-level libs

Applications and Framework

Binder IPC

Android System Services

Media Server
- AudioFlinger
- MediaPlayer Service
- Camera Service
- Other Media Services

System Server
- Search Service
- Window Manager
- Activity Manager
- Other System Services and Managers

HAL
- Camera HAL
- Audio HAL
- Graphics HAL
- Other HALs

Linux Kernel
- Camera Driver
- Audio Driver (ALSA, OSS, etc)
- Display Drivers
- Other Drivers
Adapt ANDROID
hardware/anbusu/lib

# in libanbusu.h

ssize_t anbusu_getdata(void *buf, size_t count);
ssize_t anbusu_putdata(const void *buf, size_t count);
void anbusu_clear(void);

#This is where you have to put all the complex code
(and it’s close source most of the time)
Adapt ANDROID
Dig in the framework
package android.anbusu;

public class LibAnbusu {

    public native static void clear();
    public native static String getData() throws AnbusuException;
    public native static void putData(String in) throws AnbusuException;

    static {
        System.loadLibrary("anbusu_jni");
    }
}

#LibAnbusu.java
Adapt ANDROID
hardware/anbusu/jni/

#android_anbusu_LibAnbusu.c

JNIEXPORT void JNICALL
Java_android_anbusu_LibAnbusu_putData(JNIEnv *env,
                                           jclass cls,
                                           jstring string)
{
    int ret;
    const char *buff =
    (*env)->GetStringUTFChars(env, string, NULL);
    int length = (*env)->GetStringLength(env, string);
    ret = anbusu_putdata(buff, length);
    if (ret < 0) {
        ThrowAnbusuException(env, "fail to put data");
    }
    (*env)->ReleaseStringUTFChars(env, string, buff);
}
Adapt ANDROID
Still in the framework
package android.anbusu;
/** {@hide} */
interface IAnbusuManager {
    void clear();
    String getData();
    void putData(String data);
}
### AnbusuService.java

class AnbusuService extends IAnbusuManager.Stub {

    public String getData() {
        enforceAccessPermission();
        try {
            return LibAnbusu.getData();
        } catch(AnbusuException e) {
            Slog.d(TAG, "cannot getdata");
        }
        return null;
    }

    ...

    private void enforceAccessPermission() {
    }
}
SystemServer.java

... public void run() {
... try {
    Slog.i(TAG, "Android Builder Summit Service");
    anbusuService = new AnbusuService(context);

    ServiceManager.addService(Context.PAUG_SERVICE, anbusuService);

    } catch (Throwable e) {
        reportWtf("starting Android Builder Summit Service", e);
    }
...
Adapt ANDROID
And even more ...
# AnbusuManager.java

```java
public class AnbusuManager {
    IAnbusuManager mService;

    public AnbusuManager(IAnbusuManager service) {
        mService = service;
    }

    public String getData() {
        try {
            return mService.getData();
        } catch (RemoteException e) {
            e.printStackTrace();
        }
        return null;
    }
}
```
Adapt ANDROID
Hacking ContextImpl.java

# AnbusuManager.java
class ContextImpl extends Context {

    static {
        ...

        registerService(ANBUSU_SERVICE, 
        new ServiceFetcher() {
            public Object createService(ContextImpl ctx) {
                IBinder b = ServiceManager
                .getService(ANBUSU_SERVICE);
                IAnbusuManager service = 
                IAnbusuManager.Stub.asInterface(b);
                return new AnbusuManager(service);
            }
        });
    }

    ...
}
Adapt ANDROID
make update-api ; make sdk

We are now able to
BUILD OUR SDK
and configure our IDE
Adapt ANDROID
Now the application

```java
AnbusuManager anbusuManager;

anbusuManager = (AnbusuManager) this.getSystemService(ANBUSU_SERVICE);

anbusuManager.putData("Hello Android Builder Summit!");
```
Adapt ANDROID TADAM!

Hello

hELLO
WHY DOES IT MATTERS?
Get your boss/client/girlfriend happy
WHY DOES IT MATTERS?

Make everybody happy

Think maintenance

Keep complexity in mind

And never ever forget that the maintenance work is $\binom{aC^p}{n}$ and the more you add possibilities the more you increase the maintenance costs