Google Android on the Beagleboard
Introduction to the Android API, HAL and SDK

Bill Gatliff
bgat@billgatliff.com

Freelance Embedded Systems Developer

What is Android?

"Android delivers a complete set of software for mobile devices: an operating system, middleware and key mobile applications."

-- http://android.com/about/

What is Android?

A software stack:

- ... and nothing more

(Albeit a pretty good one!)

What is Android?

A ton of new code:

- Linux kernel port to MSM (Qualcomm) chipset

- Graphics, Audio and other APIs, implementations

- Development, debugging tools

- Includes "key mobile applications"

What is Android?

Borrows heavily from existing code:

- Linux kernel for hardware abstraction

- SQLite

- libpng

- ...

```
http://source.android.com/projects
```

Configuring the BYOES Beagleboard

Steps:

- Select the Android kernel, rootfs
- Boot

On your workstation:

- Install Android development tools
- Set up USB networking

We can't do all of that today!

Configuring the BYOES Beagleboard

```
# /switchboot
***** SWITCH-UR-BOOT ****
Choose which file system to boot upon next reboot:

  1.  ESC-120 Kridner: Beagle 101
  2.  ESC-160 Van Gend/MontaVista: debugging+power
  3.  ESC-140 Fisher/RidgeRun
  4.  ESC-228 Fisher/RidgeRun
  5.  ESC-208 Gatliff: Android 1024x768
  6.  ESC-208 Gatliff: Android 800x600
  7.  ESC-180 Yau/HY-research: Bluetooth

Please enter: 5
```

Configuring the BYOES Beagleboard

```
# /switchboot

...

*** SUCCESS

The correct uImage and boot.scr have been setup.
You can press the reset button now.

#
```

Configuring the BYOES Beagleboard

Some notes:

- Keyboard and mouse work differently
- (Just ignore the mouse altogether)
- You don't have a GSM modem!

Demonstration

The Genesis of Android

Open Handset Alliance:

- Google, eBay, OMRON, PacketVideo, ...

- ASUSTeK, HTC, LG, Garmin, Motorola, ...

- Sprint Nextel, T-Mobile, ...

- ARM, Atheros, Broadcomm, Qualcomm, TI, ...

To date, more than 47 organizations

Noteworthy Features

Android uses Java:

- ... everywhere

And so will you:

- But nothing prevents native processes
- Some native interfaces are available

Noteworthy Features

Broad Java support:

- `java.io`
- `java.net`
- `java.security`
- `java.sql` ...

But only the mobile-appropriate bits!

- "Android is almost but not quite Java(tm)"

Noteworthy Features

Strong security:

- Permissions-based

- Applications sandboxed in separate VMs

- Pervasive use of Linux process model

Noteworthy Features

Built-in SQL:

- Property storage, retrieval
- Utilized by nearly all standard components
- Preferred, but not required

Specialized APIs:

- SurfaceFlinger
- AudioFlinger

Noteworthy Features

Highly-optimized Java implementation:

- "Dalvik" VM implemented by Google
- Custom bytecode format, processor model
- Register-based, not stack-based

Noteworthy Features

Why?

- "Didn't want to pay Sun" (probably untrue)
- Very memory- and performance-efficient
- Highly tuned to limitations of small hardware

Noteworthy Features

Centralized object lifetime management:

- Tied to component model

- Tied to process model

- Tied to user interface model

- Tied to security model

- ...

Noteworthy Features

No `main()` functions per se:

- "Applications" are built from components on-the-fly

- "Activities", "Intents", etc.

- "Manifests" describe what components are available

See `http://developer.android.com`

Terminology

Activity:

- A single visual user interface component

- List of menu selections, icons, checkboxes, ...

- A reusable component

Service:

- "Headless" activity component

- Background processes

Terminology

Broadcast receiver:

- Component that receives announcements
- No user interface
- May launch an Activity in response

Content provider:

- Provides application data to others
- The only way to share data

Terminology

Intent:

- Message to a component (or broadcast)
- Similar to a remote procedure call
- "Make a phone call", "the battery is low", ...

Intent filter:

- Specifies which Intents a component can handle

Terminology

Application:

- Sequence of one or more Activities
- Manifest tells which Activity to run first
- Activities might come from other applications

Process model:

- Each application is a unique Linux user
- Each application is a unique process
- Activities often in different processes

Terminology

Task stack:

- Sequences of application-centric Activity classes
- Foreground is visible to user
- BACK key returns to most-recent Activity

Terminology

In other words:

- Not the Linux concept of "application"!

Example

Display a map:

- Utilize a preexisting Activity class

- Call startActivity() to launch it

- Control returns when the map activity exits

Power Management

Obviously important!

- Can be a difficult problem to solve
- Too much model exposure is bad
- Too little is also bad

Extends the Linux device model:

- Introduces "wake locks"
- See `android.os.PowerManager`

Power Management

In a nutshell:

- Applications don't control power at all

- Applications hold "locks" on power states

- If no locks are held, Android powers down

Power Management

`PARTIAL_WAKE_LOCK`

- CPU on, screen off, keyboard off
- Cannot power down via power button

`SCREEN_DIM_WAKE_LOCK`

- CPU on, screen dim, keyboard off

Power Management

SCREEN_BRIGHT_WAKE_LOCK

- CPU on, screen bright, keyboard off

FULL_WAKE_LOCK

- CPU on, screen on, keyboard bright

Example

```
1   PowerManager pm =
2       (PowerManager) getSystemService(Context.POWER_SERVICE);
3   PowerManager.WakeLock wl =
4       pm.newWakeLock(PowerManager.SCREEN_DIM_WAKE_LOCK, "tag");
5
6   wl.acquire();
7   // ..screen will stay on during this section..
8   wl.release();
```

Audio and Video APIs

`MediaPlayer` class:

- Standard support for many data formats

- URI invokes appropriate input method

- Consistent API regardless of data source

`MediaRecorder` class:

- Support for audio recording only

- Video recording is "planned"

Example

```
1    MediaPlayer mp = new MediaPlayer();
2
3    mp.setDataSource(PATH_TO_FILE);
4    mp.prepare();
5    mp.start();
6    mp.pause();
7    mp.stop();
```

Audio and Video APIs

Surfaceflinger:

- Centralized framebuffer management
- Related to 2D h/w acceleration

Audioflinger:

- Centralized audio stream management

You don't work with these directly!

Android Build System

Usual suspects:

- `git`, `repo`
- `ant`
- `make`

Plus enhancements:

- Package management
- Application templates

Android Package System

APK files:

- Package manifests

- Classes

- Dalvik bytecodes

- Signatures, if any

`Term.apk`

Developer Tasks

Generally speaking, only two:

- Implement HAL requirements
- Implement application-specific Activity classes

As always, however:

- The devil is in the details!

Developer Tasks

HAL requirements:

- A.k.a. "Linux kernel port"

- Adapt `init.rc`

- Native code, as needed

Activity classes:

- "Writing the application"

See, it isn't so different!

System Initialization

`init.rc`

- Functionally equivalent to `inittab`

System Initialization

```
1    on init
2    sysclktz 0
3    loglevel 3
4    # setup the global environment
5        export PATH /sbin:/system/sbin:/system/bin:/system/xbin
6        export LD_LIBRARY_PATH /system/lib
7        export ANDROID_BOOTLOGO 1
8        export ANDROID_ROOT /system
9        export ANDROID_ASSETS /system/app
10       export ANDROID_DATA /data
11       export EXTERNAL_STORAGE /sdcard
```

System Initialization

```
77    on boot
78    # basic network init
79        ifup lo
80        hostname localhost
81        domainname localdomain
82
83    # set RLIMIT_NICE to allow priorities from 19 to -20
84        setrlimit 13 40 40
85
86    # Define the oom_adj values for the classes of processes that can be
87    # killed by the kernel.  These are used in ActivityManagerService.
88        setprop ro.FOREGROUND_APP_ADJ 0
89        setprop ro.VISIBLE_APP_ADJ 1
90        setprop ro.SECONDARY_SERVER_ADJ 2
```

System Initialization

```
161    ## Daemon processes to be run by init.
162    ##
163    service console /system/bin/sh
164        console
165
166    # adbd is controlled by the persist.service.adb.enable system property
167    service adbd /sbin/adbd
168        disabled
169
170    # adbd on at boot in emulator
171    on property:ro.kernel.qemu=1
172        start adbd
173
174    on property:persist.service.adb.enable=1
175        start adbd
```

Linux Kernel

Important enhancements:

- `logger`

- `binder`

- `ram_console`

- `timed_gpio`

- Double-buffered framebuffer (*)

All are staged for kernel.org release

Linux Kernel

`logger`:

- Miscdevice for logfile-like functionality

`binder`:

- Android IPC subsystem
- High performance, security-aware

Linux Kernel

`ram_console:`

- RAM-based console device
- `/proc/last_kmsg`

`timed_gpio:`

- GPIO that automagically turns itself back off

Linux Kernel

Double-buffered framebuffer:

- Added by platform support authors

- Not Android-specific, but not widely available

Building the Android Runtime

General procedure:

- Get the code
- Build it
- Install it
- :)

```
http://source.android.com/
```

Building the Android Runtime

The code:

- 2.1GB (!) of git trees
- Uses the `repo` tool to manage

Building the Android Runtime

```
# repo init -b cupcake -u
      git://android.git.kernel.org/platform/manifest.git
# repo sync

... apply tweaks ...

# make [TARGET_PRODUCT=freerunner]
```

Installing Android into a Target

Build products:

- `userdata.img`
- `ramdisk.img`
- `system.img`
- `kernel.img`

Installing Android into a Target

And also:

- `out/target/product/<name>/root`

- `out/target/product/<name>/system`

- `out/target/product/<name>/data`

Installing Android into a Target

"What's in there?"

- The Android filesystem

```
# ls root
data/           init                init.rc  sys/
default.prop    init.goldfish.rc    proc/    system/
dev/            initlogo.rle        sbin/

# ls system
app/  build.prop  fonts/      lib/    usr/
bin/  etc/        framework/  media/  xbin/
```

Installing Android into a Target

Combine into unified tree:

- ... to export over NFS, perhaps

```
# mkdir /exports/android
# cd root && tar c * | tar x -C /exports/android
# cd system && tar c * | tar x -C /exports/android
```

Installing Android into a Target

Or, of course:

- Install images into the target system as-is
- (Formats vary depending on the target)

The Android SDK

Key components:

- Compilers, other tools

- Documentation

- Examples

- Hardware emulator

- Android Debug Bridge (adb)

See `http://android.com`

Preparing the Beagleboard

Getting ready:

- Configure USB network connection
- Test `adb`

Preparing the Beagleboard

Connect OTG port:

- Configure USB networking, verify

```
$ dmesg
...
$ sudo ifconfig eth2 192.168.99.101 up
$ ping 192.168.99.100
```

Preparing the Beagleboard

Launch a shell via `adb`:

- The shell is actually on the target!

```
$ ADBHOST=192.168.99.100 adb kill-server
$ ADBHOST=192.168.99.100 adb shell
#
```

Hello, Android!

Let's start simple:

- "Hello, world!"

- Command-line tools only

```
$ activitycreator --out helloworld
      example.helloworld.HelloWorld
$ cd helloworld/
$ vi src/example/helloworld/HelloWorld.java
```

Hello, Android!

```
1    package example.helloworld;
2
3    import android.app.Activity;
4    import android.os.Bundle;
5    import android.widget.TextView;
6
7    public class HelloWorld extends Activity
8    {
9        /** Called when the activity is first created. */
10       @Override
11       public void onCreate(Bundle savedInstanceState)
12       {
13           super.onCreate(savedInstanceState);
14           TextView tv = new TextView(this);
15           tv.setText("Hello, ESC BYOE attendees!");
16           setContentView(tv);
17       }
18   }
```

Hello, Android!

Now:

- Build

- Install

- Browse to the icon, tap to launch

```
$ ant
$ ADBHOST=192.168.99.100 adb
      install bin/HelloWorld-debug.apk
```

Hello, Android!

Or:

- Launch from the shell

```
$ ADBHOST=192.168.99.100 adb shell
# am start -a android.intent.action.MAIN -n
      example.helloworld/example.helloworld.HelloWorld
```

Hello, Android!

Tidy up:

- Uninstall the application

```
$ ADBHOST=192.168.99.100 adb
      uninstall example.helloworld
```

Eclipse Android Plugin

Android Development Tool (ADT):

- Custom plugin for Eclipse IDE

Helps automate:

- Set up new Android projects

- Create new applications, components

- Debugging

Eclipse Android Plugin

Install Eclipse, then:

- Click *Help | Software Updates...*
- https://dl-ssl.google.com/android/eclipse/
- Click *Install...*

Then:

- Point Eclipse to the Android SDK directory
- *Window | Preferences | Android*

Demonstration

Declarative vs. Procedural Programming

"Programmatic" UI layout:

- UI comes directly from source code

- Manual connections between views

- Small UI changes can mean big source code changes

- Application is "brittle"

Declarative vs. Procedural Programming

A better way:

- Use a *declarative* approach

- Describe what you *want*, not how to get it

- Let the UI framework fill in the details

In Android:

- XML-based layouts, values

Hello, Android! with XML

Applied to "Hello, Android!":

- Move the layout to XML
- Move the text to a resource

Why?

- Swap `main.xml` files to change layouts
- Swap `strings.xml` files to translate
- Separate logic from presentation

Hello, Android! with XML

`res/layout/main.xml:`

- Describes the layout

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <TextView xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="fill_parent"
4     android:layout_height="fill_parent"
5     android:text="@string/hello"/>
```

Hello, Android! with XML

`res/values/strings.xml`:

- Defines the string resource

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3      <string name="hello">Welcome to Android string resources!</string>
4      <string name="app_name">Hello, Android</string>
5  </resources>
```

Hello, Android! with XML

HelloWorld class then becomes:

- "Just do what main.xml says"

```
1    package com.example.hello;
2
3    import android.app.Activity;
4    import android.os.Bundle;
5
6    public class HelloAndroid extends Activity {
7        /** Called when the activity is first created. */
8        @Override
9        public void onCreate(Bundle savedInstanceState) {
10           super.onCreate(savedInstanceState);
11           setContentView(R.layout.main);
12       }
13   }
```

Demonstration

Emulation

Android SDK includes *QEMU*:

- Complete hardware emulation
- ... without the hardware!
- Includes a Linux kernel, filesystem

... and Eclipse can't tell the difference!

"But what does all this mean?"

Why I'm excited about Android:

- New ideas on current challenges

- New developers, community

- Relatively feature-complete

- Still under active development

"But what does all this mean?"

But expecially:

- Intended, designed for community development
- (And delivers on that promise)
- Easy to get started, but still challenging

Not just a new API:

- Also an entirely new approach, context

"But what does all this mean?"

What Android seems good for:

- Open development models
- Highly-configurable systems

And obviously:

- Mobile platforms
- Touch-oriented interfaces
- Network-centric applications

"But what does all this mean?"

What Android might not be good for:

- Very low-end hardware
- Highly proprietary systems

Maybe, maybe not:

- Static systems
- Single-task systems
- No networking requirements

"But what does all this mean?"

But who knows, really? :)

Google Android on the Beagleboard
Introduction to the Android API, HAL and SDK

Bill Gatliff

bgat@billgatliff.com

Freelance Embedded Systems Developer