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
```
# /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!

Also:

- You need the Android SDK v. 1.6
Hello, Android!

Let’s start simple:

- “Hello, world!”
- Command-line tools only

```bash
$ android create project --target 2 --name Hello
   --path ./helloworld --activity HelloWorld
   --package example.HelloWorld
$ cd helloworld/
$ vi src/example/HelloWorld/HelloWorld.java
```
Hello, Android!

```java
package example.helloworld;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class HelloWorld extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView tv = new TextView(this);
        tv.setText("Hello, ESC BYOE attendees!");
        setContentView(tv);
    }
}
```
Hello, Android!

Build:

- Create a debugging-enabled package

```bash
$ ant debug
... debug:
    [echo] running zip align on final apk...
    [echo] Debug Package: bin/Hello-debug.apk

BUILD SUCCESSFUL
$
```
Hello, Android!

Create a “Virtual Device”:

- (We’ll use real hardware later)

```bash
$ android create avd --name virtual1_6 --target 2
...
Created AVD ’virtual1_6’ based on Android 1.6...
$ emulator @virtual1_6
```
Hello, Android!

Download the Package:

- Tap the icon to start it running

```
$ adb install bin/Hello-debug.apk
```
Hello, Android!

... or:

- Launch from the shell

```
$ adb shell
# am start -a android.intent.action.MAIN -n example.HelloWorld/.HelloWorld
```
“Hello, World!” on Beagle

Same as before, only:

- Redirect the debug bridge via `ADBHOST`

```
$ export ADBHOST=192.168.99.100
$ adb install bin/Hello-debug.apk
$ adb shell
# am start -a android.intent.action.MAIN -n example.HelloWorld./HelloWorld
```
“Hello, World!” on Beagle

Tidy up:

- Uninstall the application

$ adb uninstall example.HelloWorld
Success
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
- (See the instructions on developer.android.com)
The Genesis of Android

*Open Handset Alliance:*

- Google, eBay, OMRON, PacketVideo, ...
- ASUSTeK, HTC, LG, Garmin, Motorola, ...
- Sprint Nextel, T-Mobile, ...
- ARM, Atheros, Broadcom, 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)”
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
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

```xml
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:text="@string/hello"/>
```
Hello, Android! with XML

res/values/strings.xml:

- Defines the string resource

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="hello">Welcome to Android string resources!</string>
  <string name="app_name">Hello, Android</string>
</resources>
```
Hello, Android! with XML

HelloWorld class then becomes:

- “Just do what main.xml says”

```java
package com.example.hello;

import android.app.Activity;
import android.os.Bundle;

public class HelloAndroid extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```
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

```java
PowerManager pm = (PowerManager) getSystemService(Context.POWER_SERVICE);
PowerManager.WakeLock wl = pm.newWakeLock(PowerManager.SCREEN_DIM_WAKE_LOCK, "tag");
wl.acquire();
// ..screen will stay on during this section..
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

```java
MediaPlayer mp = new MediaPlayer();

mp.setDataSource(PATH_TO_FILE);
mp.prepare();
mp.start();
mp.pause();
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!
Linux Kernel

Important enhancements:

- logger
- binder
- ram_console
- timed_gpio
- Double-buffered framebuffer (*)

All are staged for/in kernel.org releases
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
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]
# make [TARGET_PRODUCT=beagleboard]
Building the Android Runtime

See also gitorious.org:

- A Beagle-specific Android repository
- Probably more up-to-date than Android proper
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

```bash
# 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)
Specific to the Beagleboard

Getting ready:

- Configure USB network connection
- Test adb
Specific to the Beagleboard

Connect OTG port:

- Configure USB networking, verify

```bash
$ dmesg
...
$ sudo ifconfig eth2 192.168.99.101 up
$ ping 192.168.99.100
```
Specific to the Beagleboard

Launch a shell via \texttt{adb}:

- The shell is actually on the target!

\begin{verbatim}
$ export ADBHOST=192.168.99.100
$ adb kill-server
$ adb shell
#
\end{verbatim}
“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 especially:

- 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? :)

Copyright © Bill Gatliiff, 2009
Google Android on the Beagleboard
Introduction to the Android API, HAL and SDK

Bill Gatliff
bgat@billgatliff.com

Freelance Embedded Systems Developer