

What else can you do with Android?

Making use of Android

Chris Simmonds

Embedded Linux Conference Europe 2010

Copyright © 2010, 2net Limited

Overview

- Creating a project
- Writing the app
- Writing native code libraries
- Other native code

Create a project

- Android build system requires a particular layout
- Use the android command or Eclipse ADT
 - Giving the target, class name (Hello), initial activity (HelloWorld) and package name (domain name):

```
android create project --target 1 --name Hello \  
--path ./helloworld --activity HelloWorld \  
--package com.example.HelloWorld
```

This is what you get

```
`-- helloworld
  |-- AndroidManifest.xml
  |-- bin
  |-- build.properties
  |-- build.xml
  |-- default.properties
  |-- libs
  |-- local.properties
  |-- res
  |   |-- drawable-hdpi
  |   |   |-- icon.png
  |   |-- drawable-ldpi
  |   |   |-- icon.png
  |   |-- drawable-mdpi
  |   |   |-- icon.png
  |   |-- layout
  |   |   |-- main.xml
  |   |-- values
  |   |   |-- strings.xml
  |-- src
  |   |-- com
  |   |   |-- example
  |   |   |   |-- HelloWorld
  |   |   |   |-- HelloWorld.java
```



Display icon in 3 resolutions

Java source code

HelloWorld.java

This code is generated for you:

```
package com.example>HelloWorld;

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

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

A hello world app

It is easy to make it print a message:

```
package com.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("Life, don't talk to me about life");
        setContentView(tv);
    }
}
```

Build

- Build using *ant*, a tool similar to *make*
 - The options are *debug* or *release*
 - Generates an Android package in [project]/bin

```
$ ant debug
Buildfile: build.xml

...
BUILD SUCCESSFUL
Total time: 1 second
```

Install

- Install on the target using adb
 - The -r option replaces any existing version

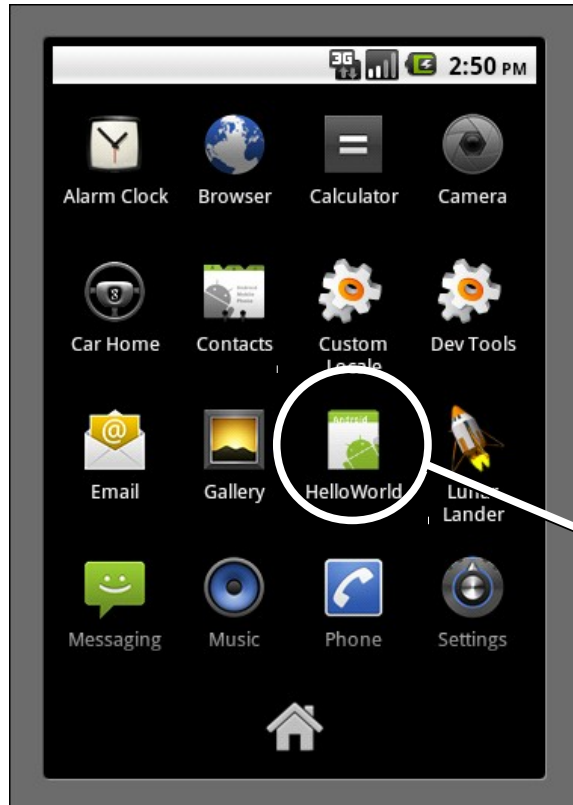
```
$ adb install -r bin/Hello-debug.apk
217 KB/s (13335 bytes in 0.059s)
  pkg: /data/local/tmp/Hello-debug.apk
Success
```

- Note: you can remove the app entirely with adb uninstall and the Java class

```
$ adb uninstall com.example.HelloWorld
Success
```


Test

This is what it looks like



Dalvik: processes and users

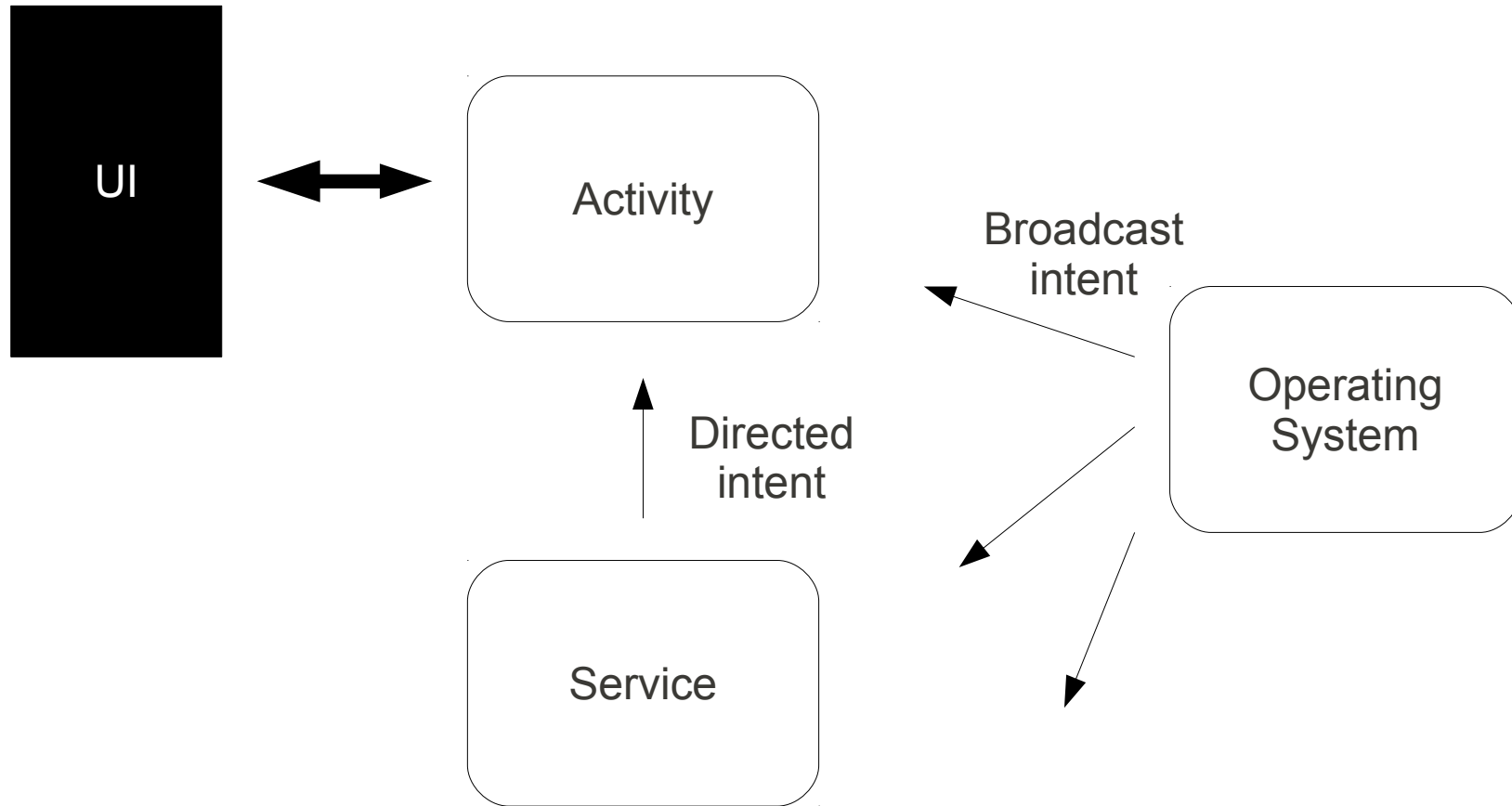
- Each app runs in separate process with a unique user name

```
# ps
USER      PID    PPID  VSIZE  RSS    WCHAN    PC         NAME
root      1      0     296    204    c009b74c 0000ca4c  S  /init

<snip>

app_9     193    32    108460 17624  ffffffff afd0eb08  S  android.process.media
app_26    203    32    119608 18072  ffffffff afd0eb08  S  com.android.mms
app_18    220    32    110136 18520  ffffffff afd0eb08  S  com.android.email
app_5     228    32    105844 16368  ffffffff afd0eb08  S  com.android.protips
app_6     259    32    106260 17624  ffffffff afd0eb08  S  com.example.HelloWorld
root     265    252     892    336    00000000 afd0d8ac  R  ps
#
```

Activities, services and intents



Activities, services and intents

- Activity: process with user interface
- Service: process without a user interface
- Intent: notification from one process to another
 - directed intent: has one specific recipient
 - broadcast intent: can be received by anyone
 - intent filter: a list of intents an activity/service is interested in

Native code

- Java Native Interface: JNI
 - allows Java code to call C/C++ functions
- The Android Native Development Kit, NDK contains the tools to create libraries of functions that are called from Java

Installing the NDK

- Download from
 - <http://developer.android.com/sdk/ndk/index.html>
- Extract to a local directory
- The next few slides show the simplest example of calling a native method: the HelloJni sample code

The C code

- This is a C function that returns a string

```
#include <string.h>
#include <jni.h>

jstring
Java_com_example_hellojni>HelloJni_stringFromJNI( JNIEnv* env,
                                                    jobject this )
{
    return (*env)->NewStringUTF(env, "Hello from JNI !");
}
```

The Java code

```
package com.example.hellojni;

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

public class HelloJni extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        TextView tv = new TextView(this);
        tv.setText( stringFromJNI() );
        setContentView(tv);
    }

    public native String  stringFromJNI();

    static {
        System.loadLibrary("hello-jni");
    }
}
```


Build

- You build the native code using the ndk-build script (a small wrapper round make):

```
$ ~/android-ndk-r4b/ndk-build
Gdbserver      : [arm-eabi-4.4.0] /home/chris/projects/android-2.2/android-
ndk-r4b/my-samples/hello-jni/libs/armeabi/gdbserver
Gdbsetup       : /home/chris/projects/android-2.2/android-ndk-r4b/my-
samples/hello-jni/libs/armeabi/gdb.setup
Gdbsetup       : + source directory /home/chris/projects/android-2.2/android-
ndk-r4b/my-samples/hello-jni/jni
Compile thumb  : hello-jni <= /home/chris/projects/android-2.2/android-ndk-
r4b/my-samples/hello-jni/jni/hello-jni.c
SharedLibrary  : libhello-jni.so
Install        : libhello-jni.so => /home/chris/projects/android-2.2/android-
ndk-r4b/my-samples/hello-jni/libs/armeabi
```

Incorporate into a project

- The ndk sample code does not include all the project files
- You need to create a project with the appropriate name and Java class
- Build and install as before
 - details in the handout in section 3.

Installed files

Three files are installed this time

The package:

```
/data/app/com.example.HelloJni.apk
```

The dex (compiled Java) file:

```
/data/dalvik-cache/data@app@com.example.HelloJni.apk@classes.dex
```

The library:

```
/data/data/com.example.HelloJni/lib/libhello-jni.so
```

Officially-sanctioned native libraries

- These libraries form a stable API that should be on all Android platforms:

Library	Header	API level	Notes
libc	stdlib.h, etc	3	“Bionic” C library
libpthread	pthread.h	3	Simplified threads
libm	math.h	3	Maths library
libstdc++	cstdint, etc	3	Minimal C++. No exceptions or RTTI
liblog	android/log.h	3	Logging
libz	zlib.h	3	Compression
libdl	dldfcn.h	3	Dynamic linker library
libGLESv1	GLES/gl.h	4	OpenGL ES 1.x rendering
libGLESv2	GLES2/gl2.h	5	OpenGL ES 2.0 rendering
libjnigraphics	android/bitmap.h	8	Access Java bitmap objects

Adding your own libraries

- Should you want to use a library not on the official list, then
- It may be part of the build already
 - e.g. libsqlite, libjpeg
- Otherwise you will have to cross-compile using the Android tool chain
 - Outside the scope of this presentation

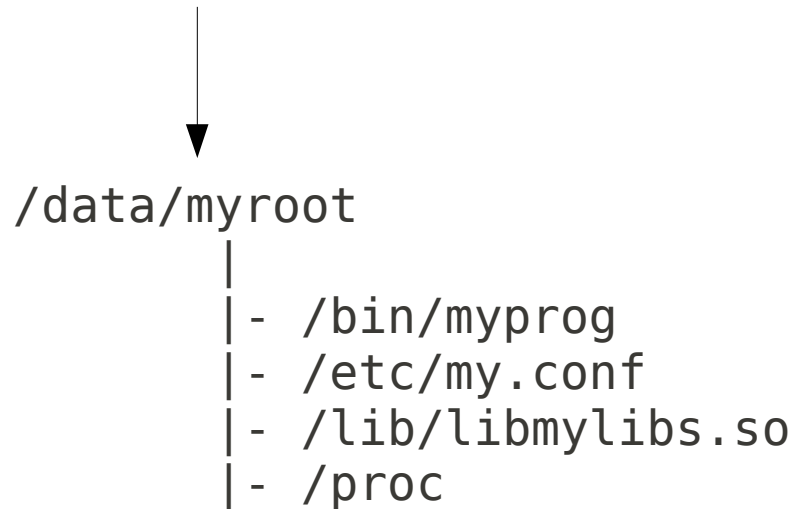
Integrating non-Android C/C++ code

- For example some kind of middle-ware
- Cross-compiling for Android is hard because
 - bionic is not a standard libc
 - limited libstdc++
 - limited selection of other libraries
- Two other options
 - static link - no library dependencies
 - chroot - create your own root for your program

Using a chroot

To launch *myprog* with root = /data/myroot

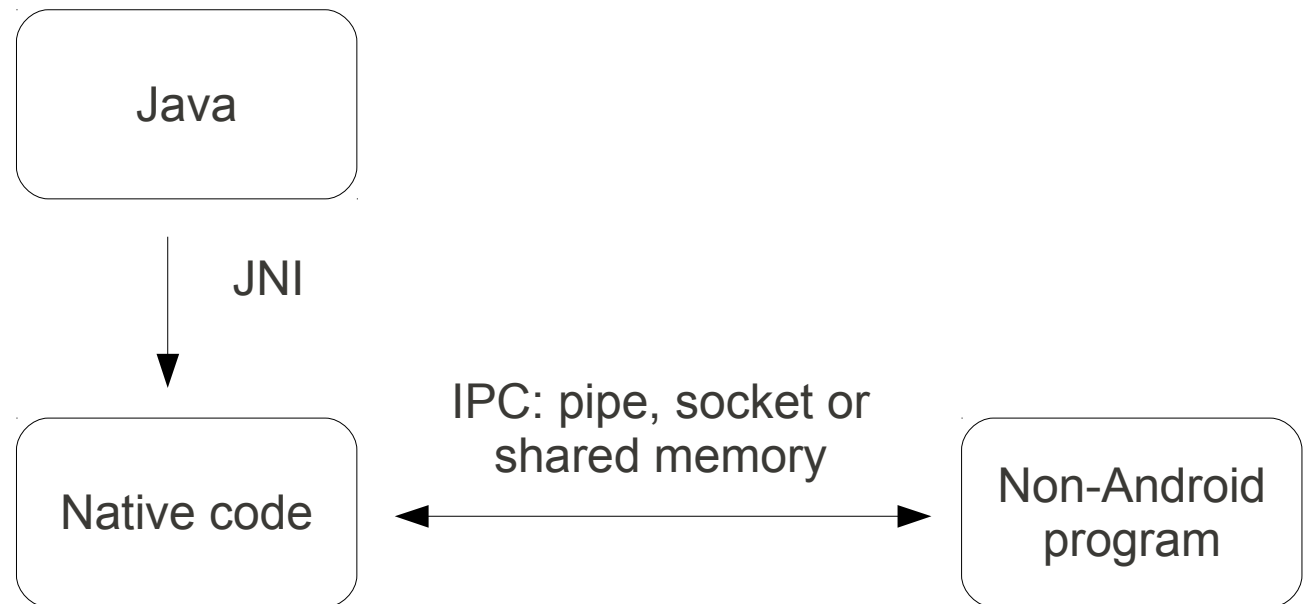
```
chroot /data/myroot /bin/myprog
```



Note that the `chroot` command is not in Android. You could use `busybox` or write your own simplified `chroot`

Communicating with non-Android code

- Non-Android code cannot communicate with Java code via JNI
- Have to use another form of IPC:



Summary

- Android applications are written in Java which is compiled into a Dalvik executable and packaged for the target
- The principle event mechanism is the intent
 - Activities and services can listen for intents
- Java code can call C/C++ functions in shared libraries by using the NDK