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):

```bash
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
```
HelloWorld.java

This code is generated for you:

```java
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:

```java
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
#```

Making use of Android
Activities, services and intents

UI

Activity

Directed intent

Service

Broadcast intent

Operating System

Making use of Android
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
- 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

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

jstring
Java_com_example_hellojni_HelloJni_stringFromJNI( JNIEnv* env,
                                          jobject thiz )
{
    return (*env)->NewStringUTF(env, "Hello from JNI !");
}
```
The Java code

```java
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`):

```sh
$ ~/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:

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

```
Java

JNI

Native code

IPC: pipe, socket or shared memory

Non-Android program
```
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.