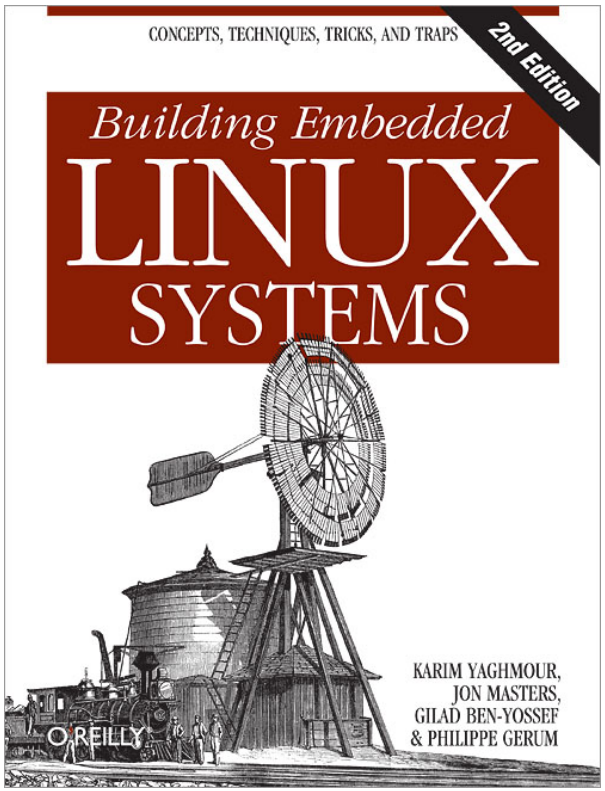
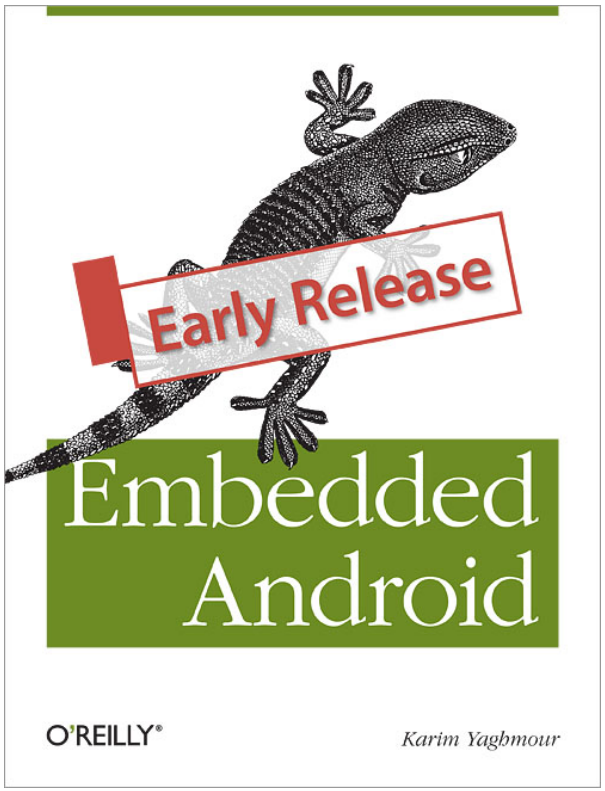
Class Structure and Presentation

Opersys inc.

August 12, 2012

About me





- Introduced "Linux Trace Toolkit" in late '90s
- Originated Adeos and relayfs (kernel/relay.c)

About Android

- Huge
- Stealthy
- Fast moving

Goals

- Provide an in-depth overview of Android from a developer's perspective
- Enable you to create full-fledged Android apps
- Give you a hands-on experience with Android development

Audience

- Developers seeking to:
  - Develop Android apps
  - Deliver mobile versions of web/enterprise apps
  - Port apps to Android
- Requirements: Java

Topcis I

- Class Structure and Presentation
- Introduction to Android Development
- Application Fundamentals
- User Experience
- Application Resources
- Intents
- Data Storage
- Content Providers
- Security and Permissions
- Manifest File
- Binder IPC
- REST-based apps

Topcis II

- Administration
- Packaging and Distribution
- WebKit-based Apps
- App Widgets
- Telephony/SIP
- Audio/Video
- Sensors
- Graphics
- Native development
- Internals
- Location and Maps
- Bluetooth
- NFC

Courseware

- These slides are based on Google's own documentation
- Exercises
- **Google's documentation**

Introduction to Android Development

Opersys inc.

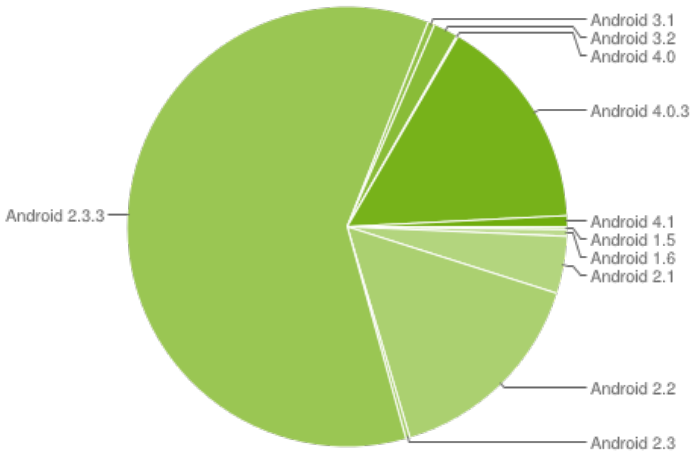
August 12, 2012

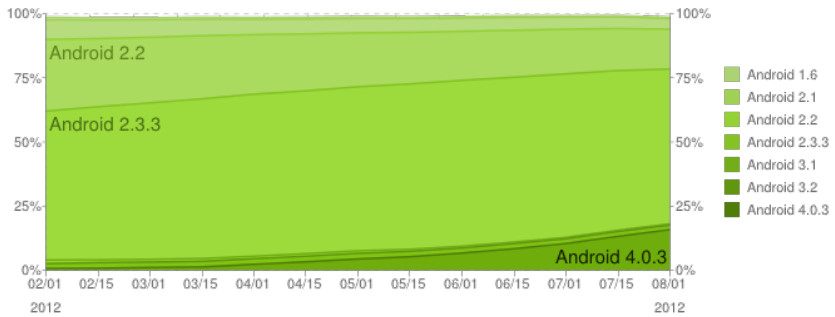
What is and isn't Android

- IS:
  - Tailored for touch-based app UX
  - App marketplace
  - Custom OS distribution
  - Complete and coherent app development API
  - Java based
  - Fully-integrated development suite
  - Very well documented development platform
  - Growing development community
- ISN'T:
  - Traditional Java (SE/ME/foo or otherwise)
  - Traditional "application" development model
  - Traditional Embedded Linux system

Android, the numbers

- Android is currently **on fire**
- 850k phone activations per day
- 400k apps (vs. 585k for Apple's app store)
- 50% total US smartphone subscribers
- ...





Where does Android come from

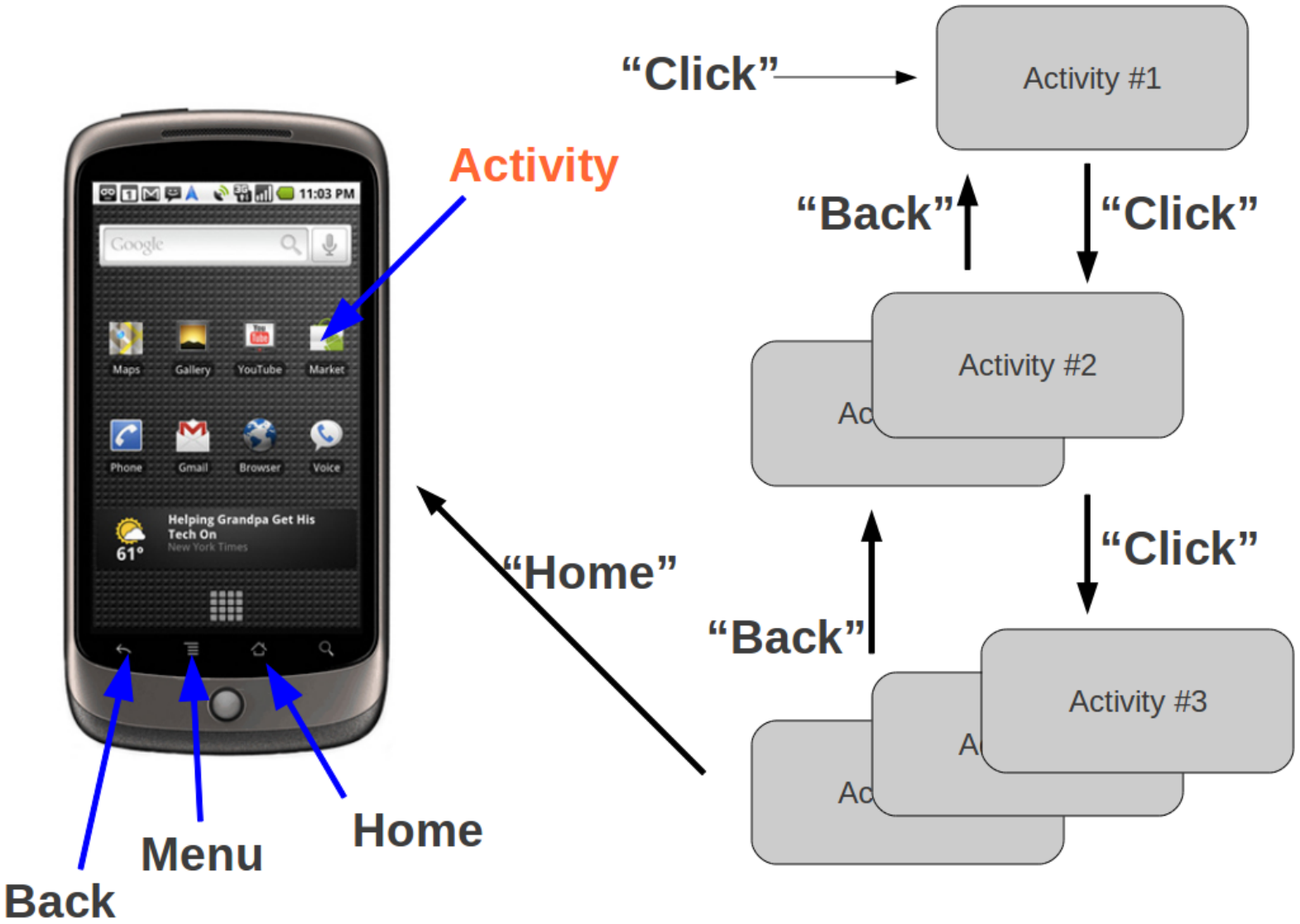
- 2002:
  - Sergey Brin and Larry Page started using Sidekick smartphone
  - Sidedick one of 1st smartphones integrating web, IM, mail, etc.
  - Sidedick was made by Danger inc., co-founded by Andy Rubin (CEO)
  - Brin/Page met Rubin at Stanf. talk he gave on Sidekick's development
  - Google was default search engine on Sidekick
- 2004:
  - Despite cult following, Sidekick wasn't making $
  - Danger inc. board decided to replace Rubin
  - Rubin left. Got seed $. Started Android inc. Started looking for VCs.
  - Goal: Open mobile hand-set platform
- 2005 - July:
  - Got bought by Google for undisclosed sum :)
- 2007 - November:
  - Open Handset Alliance announced along with Android
- 2008 - September:
  - Android 1.0 is released

- 2009 - Feb.: Android 1.1
- 2009 - Apr.: Android 1.5 / Cupcake
- 2009 - Sept.: Android 1.6 / Donut
- 2009 - Oct.: Android 2.0/2.1 / Eclair
- 2010 - May: Android 2.2 / Froyo
- 2010 - Dec.: Android 2.3 / Gingerbread
- 2011 - Jan : Android 3.0 / Honeycomb - Tablet-optimized
- 2011 - May : Android 3.1
- 2011 - Nov : Android 4.0 / Ice-Cream Sandwich - merge of Gingerbread and Honeycomb
- 2012 - Jun : Android 4.1 / Jelly Bean - Speed enhancements

App model (vs. “classic” applications)

- No single entry point (No main() !?!?)
- Unlike Windows or Unix API/semantics in many ways
- Processes and apps will be killed at random: developer must code accordingly
- UI disintermediated from app “brains”
- Apps are isolated, very
- Behavior predicated on low-memory conditions

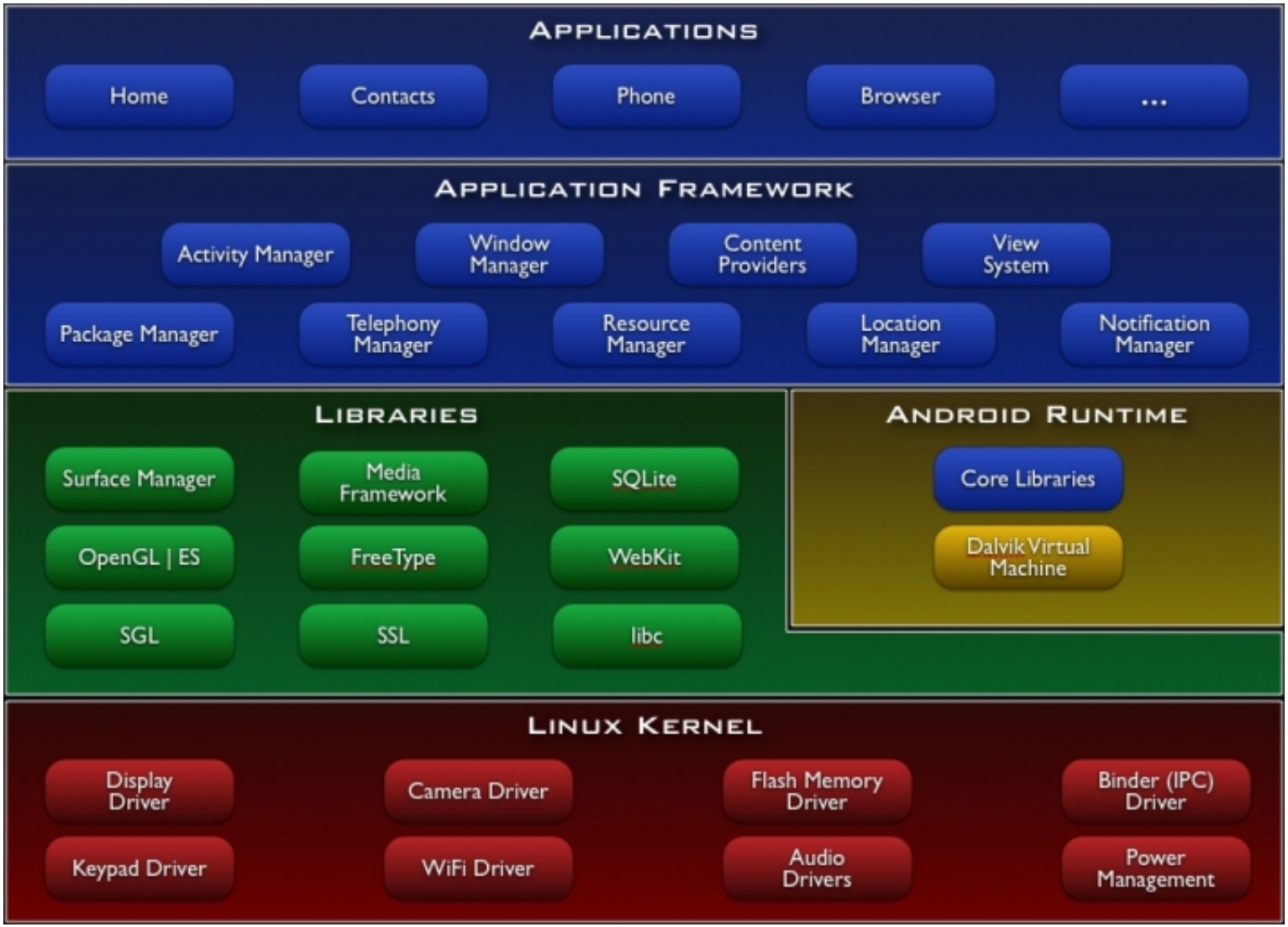
User experience



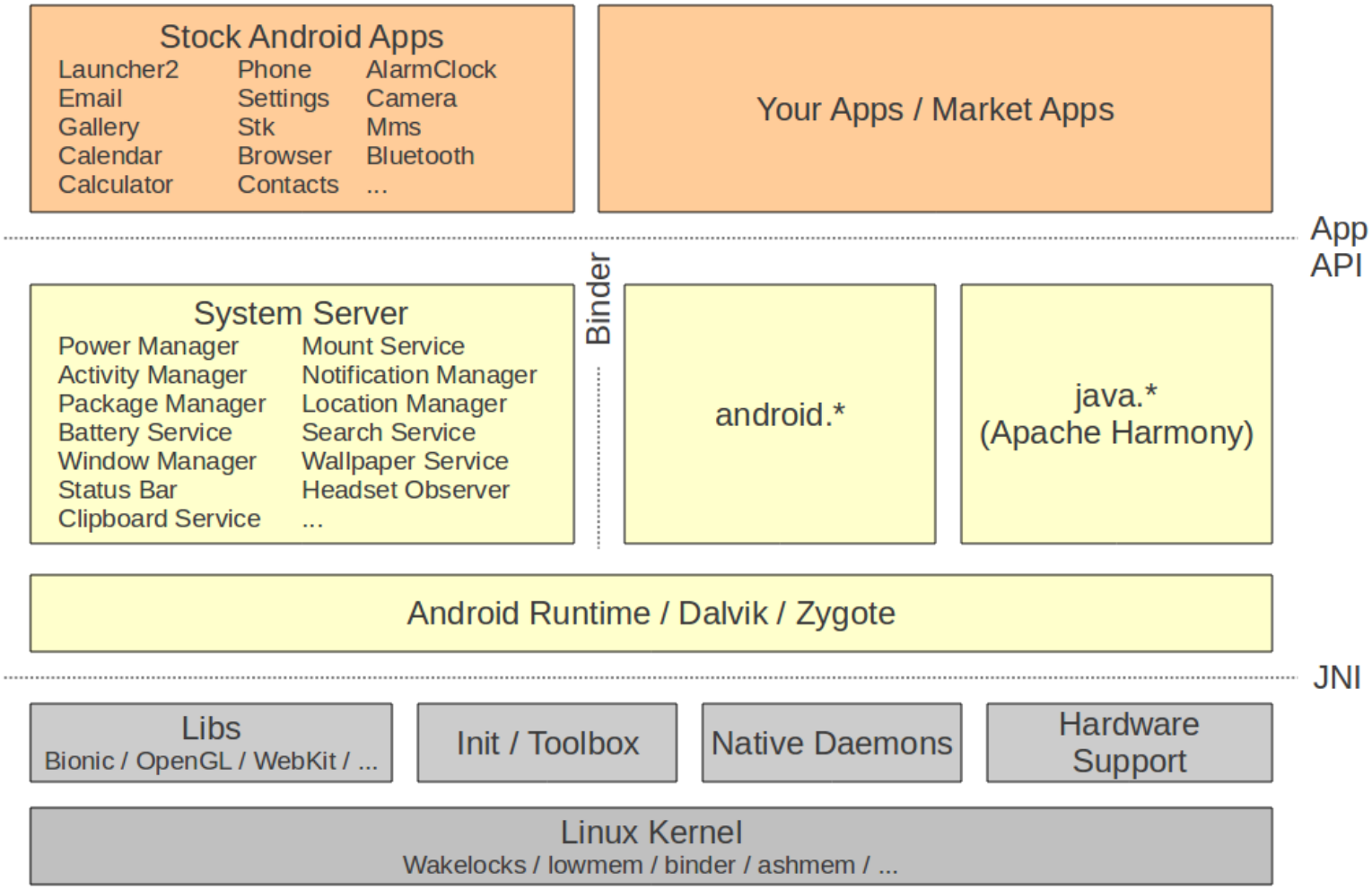
Features - as advertized by Google

- Application framework enabling reuse and replacement of components
- Dalvik virtual machine optimized for mobile devices
- Integrated browser based on the open source WebKit engine
- Optimized graphics powered by a custom 2D graphics library; 3D graphics based on the OpenGL ES 1.0 specification (hardware acceleration optional)
- SQLite for structured data storage
- Media support for common audio, video, and still image formats (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- GSM Telephony (hardware dependent)
- Bluetooth, EDGE, 3G, and WiFi (hardware dependent)
- Camera, GPS, compass, and accelerometer (hardware dependent)
- Rich development environment including a device emulator, tools for debugging, memory and performance profiling, and a plugin for the Eclipse IDE

Internal architecture - Google's version



Internal architecture - the "real thing"



SDK set up and update

- What's in the SDK?
  - Android-specific tools
  - Eclipse plugin
  - QEMU-based emulator
- Getting the SDK
  http://developer.android.com/sdk/index.html
- Prerequisites:
  - Windows, Mac or Linux
  - Eclipse 3.4 or later - (highly recommended)
  - JDK 5 or 6 (gcj won't do)

SDK set up and update

1. Make sure prerequisites are installed
2. Install SDK
3. Install ADT plugin for Eclipse
4. Use “Android SDK and AVD Manager” (**android** cli tool) to install platform support

Basic debugging tricks

- Logging - Use LogCat to view (either using 'adb logcat' or Ecplise):

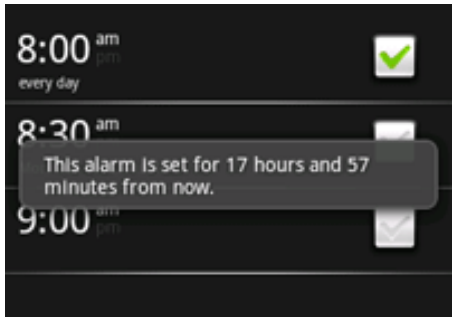
```
import android.util.Log;
...
Log.d(DTAG, "onConfigurationChanged() called");
```

- Log output:

```
D/PhoneApp(  229): updateProximitySensorMode: state = IDLE
D/PhoneApp(  229): updateProximitySensorMode: lock already released.
W/dalvikvm(  824): threadid=1: thread exiting with uncaught exception (group=0
E/AndroidRuntime(  824): FATAL EXCEPTION: main
```

- Toast messages - show up device UI:

```
import android.widget.Toast;
...
Toast.makeText(this, "onConfigurationChanged()", Toast.LENGTH_SHORT).show();
```



Development and debugging tools

- android - manage AVDs and SDK components
- apkbuilder - creating .apk packages
- dx - converting .jar to .dex
- adb - debug bridge
- ddms - Dalvik debug monitor
- monkey - test UI randomly
- traceview - view app's execution logs
- logcat - view system logs
- ...

Alternative development frameworks

- PhoneGap (FOSS):
  - Apps written in JavaScript/HTML/CSS
  - Apps run in browser widget
  - Supports: iOS, iPad, Android, Palm, Symbian, BB
  - http://www.phonegap.com/
- Appcelerator Titanium (FOSS):
  - Apps written in JavaScript/HTML/CSS
  - Apps compilled to native using SDK
  - Supports: Android, iOS, Windows, Mac, Linux
  - http://www.appcelerator.com/
- MonoDroid (proprietary):
  - Apps written in C#/.NET
  - Mono runtime runs side-by-side w/ Dalvik
  - http://monodroid.net/
- SL4A - Scripting Languages for Android / alpha-quality:
  - Python, Perl, JRuby, Lua, BeanShell, JavaScript, Tcl, shell ..
  - https://code.google.com/p/android-scripting/
- ...

Alternative app marketplaces

- App-manager has to be root to install apps
- User can circumvent by allowing install from “Unknown sources” :(
- Manufacturer and/or carrier can factory-install other “market”
- Verizon V Cast
- Amazon
- B&N NOOK Apps
- F-Droid: FOSS app backend

Application Fundamentals

Opersys inc.

August 12, 2012

Introduction

- All app contents packaged into .apk:
  - Compiled Java
  - Resources
  - Data
  - Native code
  - etc.
- .apk generated by $\text{aapt}_{\text{tool}}$
- .apk is what is downloaded by users to their device
- 1 apk = 1 app

- Apps live in isolated worlds:
  - Each App is assigned its own UID:
    - Separate permissions (ex: /home/foo vs. /home/bar)
    - No files shared
  - Every App is its own separate Linux process:
    - Process startup/shutdown is automagic/on-demand
    - No single point of failure
  - Each process has its own Dalvik VM:
    - No inter-app code interference
    - No direct cross-app calls
- Exceptions:
  - Can arrange for 2 apps = 1 UID (shared files)
  - Can arrange for 2 apps of 1 UID = 1 Process

Application Components

- 1 App = N Components
- Apps can use components of other applications
- App processes are automagically started whenever any part is needed
- Ergo: N entry points, !1, and !main()
- Components:
  - Activities
  - Services
  - Broadcast Receivers
  - Content Providers

Activities

- Visual interface for a **single** user interaction
- Activities are independent from one another
- `public class Foo extends Activity { ... }`
- 1 Activity = 1 default window, usually full-screen
- Visual content = hierarchy of views ($\text{View}_{\text{class}}$):
  - ex.: containers, buttons, scroll bars, etc.
- Activity.setContentView() = set root $\text{View}_{\text{class}}$

Services

- Background service
- Runs indefinitely
- Talk to service = bind to it
- `public class Foo extends Service { ... }`
- Runs in main app process
- Ergo: use accept()/fork() semantics
- Remote calls run on thread pool

Broadcast Receivers

- Receive and react to broadcast announcement
- Usually system event:
  - Ex.: low bat, new pic, timezone change
- Apps can also initiate broadcasts
- `public class Foo extends BroadcastReceiver { ... }`
- 1 App can have N Broadcast Receivers
- No UI, but can start Activity or send Notifications

Content Providers

- Makes data available to other apps
- Data can be stored in FS or SQLite
- `public class Foo extends ContentProvider { ... }`
- Apps use $\text{ContentResolver}_{\text{object}}$ to talk to Content Provider
- All IPC is transparent when using $\text{ContentResolver}_{\text{object}}$

Intents

- Intent = asynchronous message w/ or w/o designated target
- Like a polymorphic Unix signal, but w/o required target
- Intents "payload" held in $\text{Intent}_{\text{object}}$
- Intent Filters specified in Manifest file

Components activation and shut down

- Activity
- Service
- Content Provider

Activity

- Activated through passing Intent to:
  - Context.startActivity()
  - Activity.startActivityForResult()
- Activity's onCreate() doesn't provide the Intent
- Use getIntent() to look at initial Intent
- Subsequent Intents sent to onNewIntent() callback
- Intent results sent to onActivityResult() callback
- Shut down:
  - Self: finish()
  - Other Activity started w/ startActivityForResult(): finishActivity()
  - System

Service

- Activated through:
  - Passing Intent to Context.startService()
  - Call to Context.bindService()
- Both generate an onCreate() callback
- Context.startService() generates onStart() callback:
  - Takes Intent as parameter
- Context.bindService() generates onBind() callback:
  - Use RPC thereafter
- Shut down:
  - Self: stopSelf()
  - Other Component: Context.stopService()
  - System

Broadcast Receiver

- Send Intent through:
  - Context.sendBroadcast()
  - Context.sendOrderedBroadcast()
  - Context.sendStickyBroadcast()
- Trigger onReceive() callback
- Active while it's responding to broadcast message
- Can be shut down by system

Content Provider

- Activated through request from ContentResolver
- Active while it's responding to ContentResolver request
- Can be shut down by system

The Manifest file

- Informs system about app's components
- XML format
- Always called AndroidManifest.xml
- Activity = `<activity>` ... static
- Service = `<service>` ... static
- Broadcast Receiver:
  - Static = `<receiver>`
  - Dynamic = Context.registerReceiver()
- Content Provider = `<provider>` ... static

```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
    <application . . . >
        <activity android:name="com.example.project.FreneticAc
                  android:icon="@drawable/small_pic.png"
                  android:label="@string/freneticLabel"
                  . . .  >
        </activity>
        . . .
    </application>
</manifest>
```

- Two types of Intents:
  - w/ explicitly named targets
  - w/o explicitly named targets
- When no target is named, system uses Filters to locate best fit
- If Intent isn't named in Filter, activation only on explicit naming
- Intent Filter for Launcher icon Activity:
  - action: android.intent.action.MAIN
  - category: android.intent.category.LAUNCHER

```
. . .
<intent-filter . . . >
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" /
</intent-filter>
<intent-filter . . . >
    <action android:name="com.example.project.BOUNCE" />
    <data android:mimeType="image/jpeg" />
    <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
. . .
```

Activities and Tasks

- Task = group of related Activities arranged in a stack
  - This is what the user experiences as being a "classic" "application"
- Root Activity = Activity that began the Task (typically from Launcher)
- New Activity = push new Activity on stack
  - Current Activity = new Activity
- BACK key = pop current Activity
  - Current Activity = previous Activity
- Activity stack is **never** rearranged: just push + pop
- There is **no** "task" construct: no class nor Manifest entry

- Default: Tasks are "swapped" to/from foreground in their entirety
- Default can be overriden
- Activity behavior within Task depends on:
  1. Flags set in $\text{Intent}_{\text{object}}$ that started Activity
  2. Attributes set in <activity> element in Manifest
- IOW, requester AND respondent have a say in behavior

- Intent Flags:

  `FLAG_ACTIVITY_NEW_TASK`
  `FLAG_ACTIVITY_CLEAR_TOP`
  `FLAG_ACTIVITY_RESET_TASK_IF_NEEDED`
  `FLAG_ACTIVITY_SINGLE_TOP`

- `<activity>` Attributes:

  `taskAffinity`
  `launchMode`
  `allowTaskReparenting`
  `clearTaskOnLaunch`
  `alwaysRetainTaskState`
  `finishOnTaskLaunch`

Affinities and new Tasks

- Default: Activities in same app prefer being in same Task
- However, each Activity can have an individual $\text{taskAffinity}_{\text{attribute}}$ set within the parent `<activity>` element.
- Therefore:
  - Activities in different apps can share same Affinity
  - Activities in same app can have different Affinities
- Affinity relevant when:
  - Activity-launching Intent has $\text{FLAG\_ACTIVITY\_NEW\_TASK}_{\text{flag}}$ set:
    - Launch new Task for Activity, if none with corresponding taskAffinity exists
  - Activity has $\text{allowTaskReparenting}_{\text{attribute}}$ set to "true":
    - Activity will migrate from its current Task to one it has Affinity to if comes to the fore
- Typically, use different Affinities if .apk has multiple apps (Launcher icons)

Launch modes

- launchMode$_{attribute}$ in `<activity>` element:
  - `"standard" (the default mode)`
  - `"singleTop"`
  - `"singleTask" (use only for Launcher-displayed apps)`
  - `"singleInstance" (use only for Launcher-displayed apps)`
- Which Task for Intent-launched Activity?
  - "standard"—"singleTop" = Task that originated Intent, except if FLAG_ACTIVITY_NEW_TASK is set
  - "singleTask"—"singleInstance" = separate root Task, not Intent-originator
- Can there be multiple Activity instances?
  - "standard"—"singleTop": Yes
  - "singleTask"—"singleInstance": No
- Can Activity instance have other Activities in Task?
  - "standard"—"singleTop"—"singleTask": Yes
  - "singleInstance": No. This Activity is alone in its Task, always.

- Will new class instance be launched to handle new Intent?
  - "standard": Yes
  - "singleTop":
    - If none on top existing stack: Yes
    - If exists instance on top of existing stack: No
  - "singleTask"—"singleInstance": No
    - If Activity of "singleTask" is not on top, Intent dropped but Task comes to the fore.
- BACK key:
  - If new Activity created to handle new Intent: BACK = previous Activity
  - If existing Activity handles new Intent: BACK != previous Activity

Clearing the stack and Task relaunch

- Default:
  - If user leaves task for long time, system clears task of all Activities except root.
- Activity Attributes to modify default:
  - alwaysRetainTaskState$_{\text{attribute}}$:
    - If set to "true" for root Activity, system doesn't clear Task
  - clearTaskOnLaunch$_{\text{attribute}}$:
    - If set to "true" for root Activity, system **always** clears Task
  - finishOnTaskLaunch$_{\text{attribute}}$:
    - If set to "true" for **any** Activity, Activity disappears on next launch
- Intent Flags that modify default:
  - FLAG_ACTIVITY_CLEAR_TOP:
    - If target Task already has Activity instance, all Activities above it are cleared

Starting Tasks

- To display Activity in the main app Launcher, set `<activity>` Intent Filters in Manifest:
  - action: android.intent.action.MAIN
  - category: android.intent.category.LAUNCHER
- Activities that should almost always have a Launcher icon:
  - All "singleTask" and "singleInstance" Activities
  - Activities expecting Intent w/ FLAG_ACTIVITY_NEW_TASK Flag

Processes and threads

- 1st time Components need to run: System starts Linux process
- Default: **all Components of an app run in single process thread**
- Defaults can be overriden:
  - Run Components in other processes
  - Spawn additional threads for any process

Processes

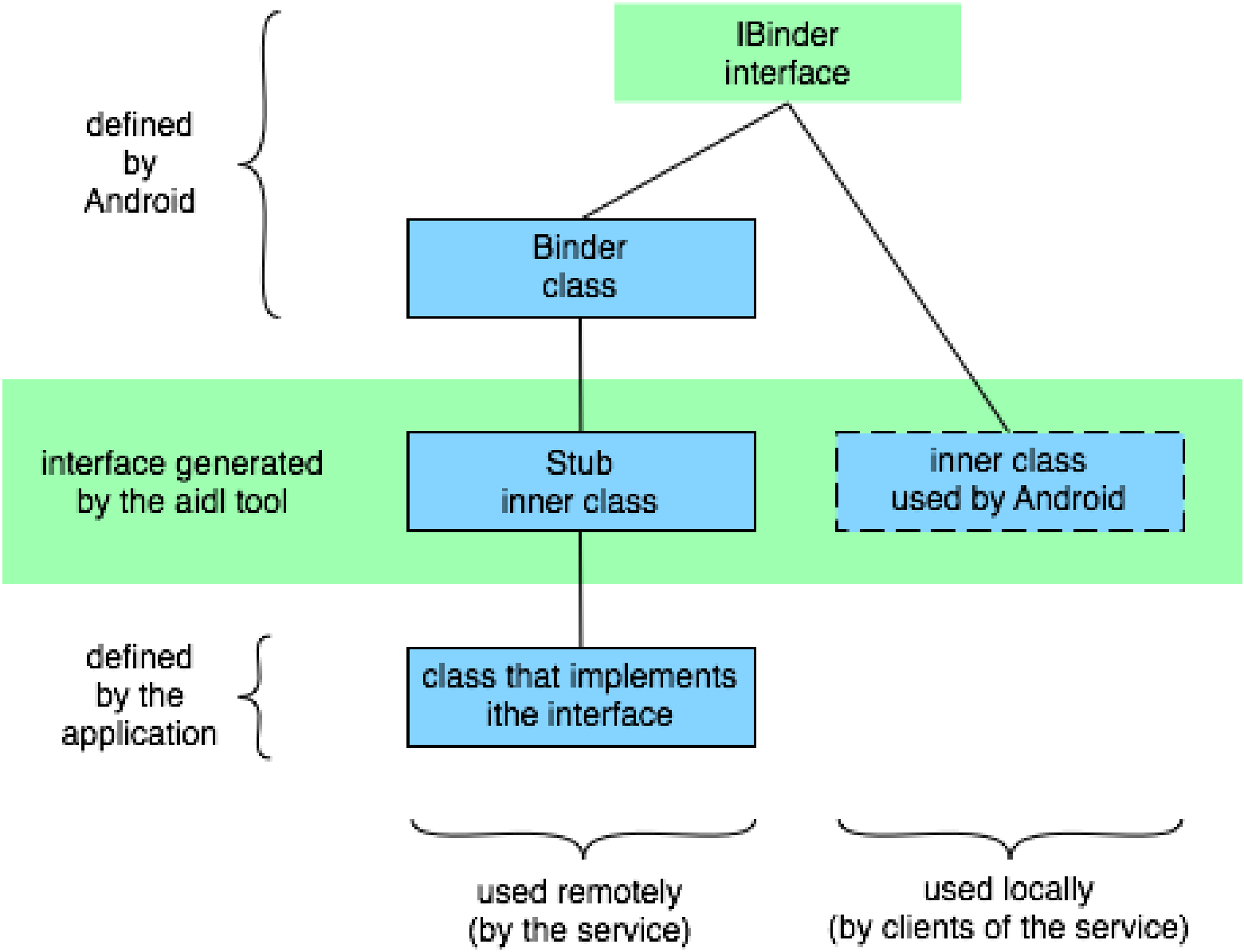
- Default: all callbacks to any app Component are issued to the main process thread
- `<activity>`—`<service>`—`<recipient>`—`<provider>` have $\text{process}_{\text{attribute}}$ to override default:
  - String name of process (default: same as app package name)
  - If starts w/ ":", process "private" to app is created for Component
  - If starts w/ lowercase, Component runs in global process of that name if permissions allow
- Do **NOT** perform blocking/long operations in main process thread:
  - Spawn threads instead
- **Process termination/restart is at system's discretion**
- Therefore:
  - Must manage Component Lifecycle

Threads

- Create using the regular Java $\text{Thread}_{\text{object}}$
- Android API provides thread helper classes:
  - Looper: for running a message loop with a thread
  - Handler: for processing messages
  - HandlerThread: for setting up a thread with a message loop

Remote procedure calls

- Android RPCs = Binder mechanism
- Binder is a low-level functionality, not used as-is
- Instead: must define interface using Interface Definition Language (IDL)
- IDL fed to $\text{aidl}_{\text{tool}}$ to generate Java interface definitions


IBinder
interface
defined
by
Android
Binder
class
interface generated
by the aidl tool
Stub
inner class
inner class
used by Android
defined
by the
application
class that implements
ithe interface
used remotely
(by the service)
used locally
(by clients of the service)

- Code generated by $\text{aidl}_{\text{tool}}$ takes care of all IPC details
- Typically, the invoked party is a Service
- Client:
  - Implements onServiceConnected() callback, receives IBinder object
  - Implements onServiceDisconnected() callback
  - Calls bindService() to connect to Service
- Service:
  - Subclasses $\text{aidl}_{\text{tool}}$-generated Stub to implement interface
  - onBind() callback decides whether or not to accept connection based on Intent
  - onBind() returns instance of Stub subclass
- **All RPC calls are synchronous**

Thread-safe methods

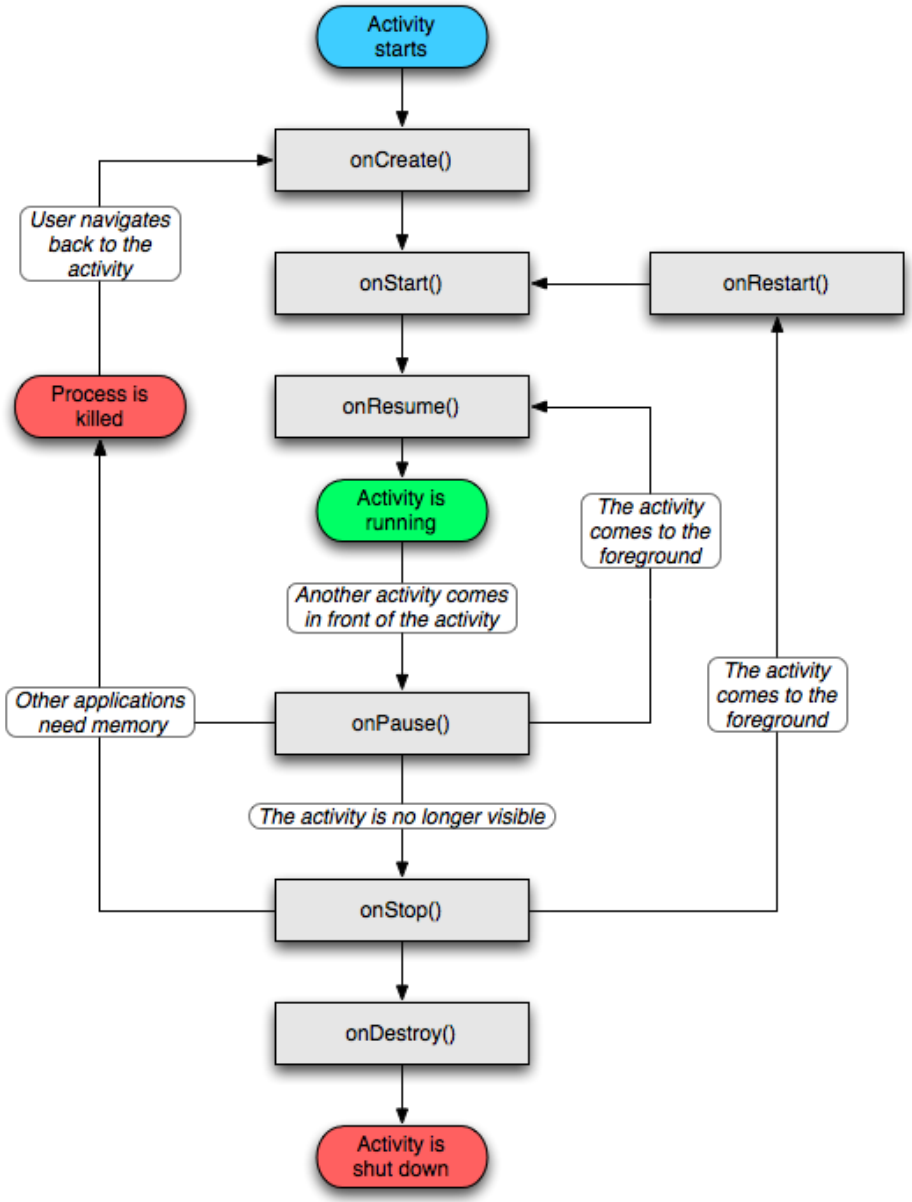
- Remote callbacks don't run in Component's process main thread
- Remote callbacks run from a thread pool:
  - Methods exposed by an IBinder Stub subclass
  - ContentProvider methods (query(), insert(), delete(), update(), getType())
- Since many clients can call the same methods: methods exposed remotely **must** be thread safe:
  - 'synchronized' keyword
  - java.lang.Thread
  - java.util.concurrent*
  - ...

Component Lifecycles

- System automagically starts/stops/kills processes
- System triggers Lifecycle callbacks when relevant
- Ergo: Must manage Component Lifecycle
- Some Components are more complex to manage than others

Activity Lifecycle

- Activity's states:
  - Resumed (running): top of Task's Activity stack
  - Paused: lost focus but still visible
  - Stopped: completely hidden to user
- Paused or stopped Activities are garbage-collectable


Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
Other applications need memory
onPause()
The activity comes to the foreground
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

- Lifecycle:
  - Entire lifetime: between onCreate() and onDestroy()
  - Visible lifetime: between onStart() and onStop()
  - Foreground lifetime: between onResume() and onPause()
- Must implement onCreate()
- Should always call superclass first:

```
protected void onPause() {
    super.onPause();
    . . .
}
```

- "Kill-ability": whether system can kill process after callback
- Unkillable: onCreate()—onStart()—onRestart()—onResume()
- Killable: onPause()—onStop()—onDestroy()
- onPause() is **last call guaranteed** to run to its full:
  - Commit to storage here or forever remain silent if you loose data

Coordinating Activities

- Activity switch call sequence:
  - Current Activity onPause()
  - New Activity onCreate(), onStart() and onResume()
  - Current Activity's onStop(), if no longer visible

Saving Activity state

- User doesn't care about system's stop/restart magic
- Must **transparently** save AND restore state
- onSaveInstanceState() called before onPause():
  - Gets $\text{Bundle}_{\text{object}}$ to save name-value pairs
- On restart, $\text{Bundle}_{\text{object}}$ passed to (use either/or or both):
  - onCreate()
  - onRestoreInstanceState(), called after onStart()
- onSaveInstanceState()—onRestoreInstanceState() not part of Lifecycle:
  - Calling depends on state saving's usefulness
  - Ex.: BACK key will result in onPause() but no onSaveInstanceState()

System configuration changes

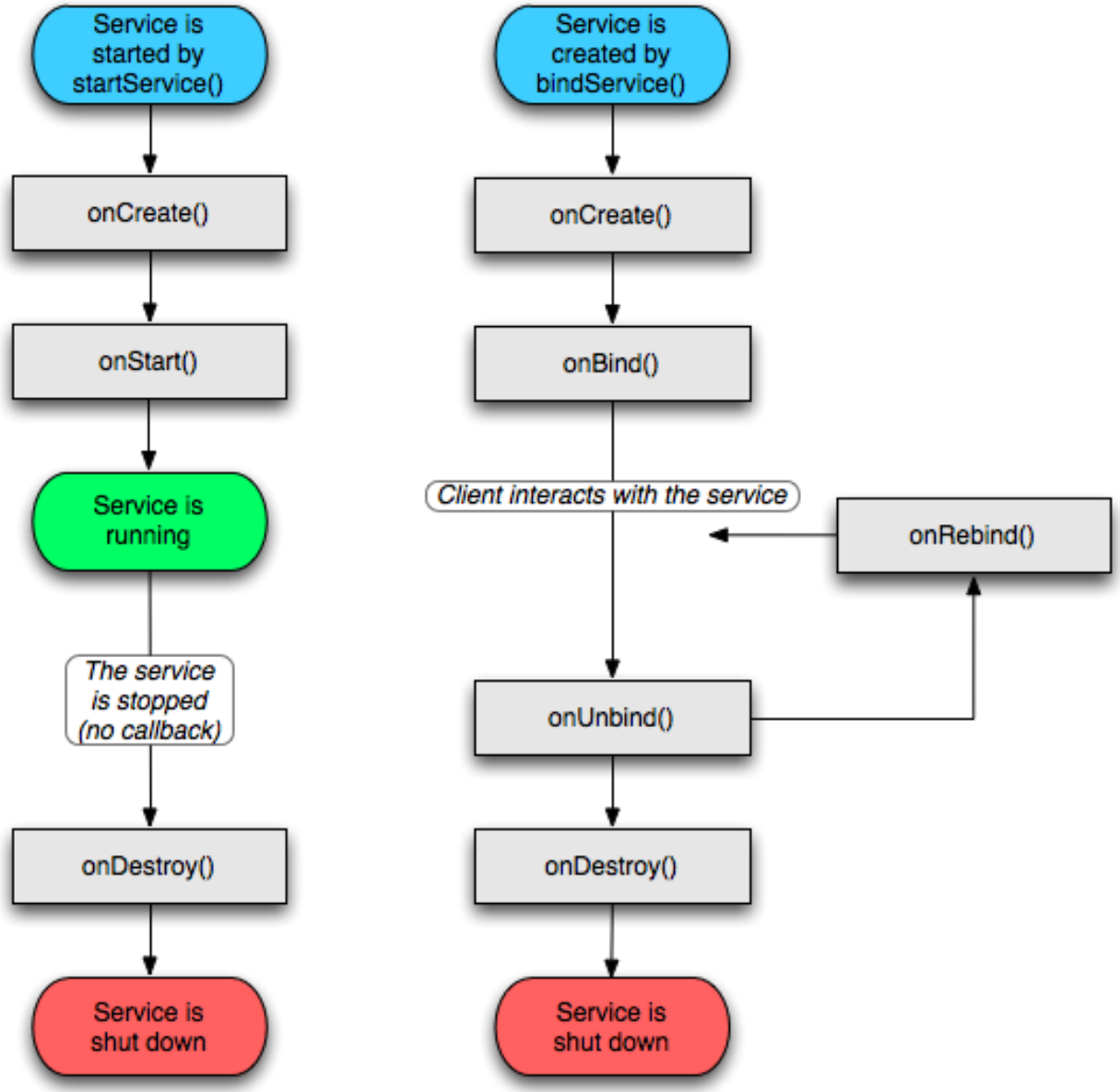
- If config changes, system will stop and restart Activity w/ new resources
- More on resources later
- Screen orientation change = config change
- Either let system restart Activity and save/restore its state:
  - Bundle, for small amounts of data
  - Object, for large/non-trivial data (sockets, bitmaps, etc.)
- Or handle the config change yourself

- Bundle saving/restoring: as explained above
- Object saving/restoring
  - Override onRetainNonConfigurationInstance() callback
  - Return properly populated object when invoked
  - Use getLastNonConfigurationInstance() in onCreate() to get it back
- Never include any objects tied to the possibly-dying Activity #LEAKS
- onRetainNonConfigurationInstance() called between:
  - onStop()
  - onDestroy()
- This is after onPause() and therefore **NOT guaranteed**
- Called **only** in the case of configuration changes, not stop/restart

- Manually handling config change:
  - Use configChanges$_{attribute}$ in Manifest to list changes handled: orientation, keyboardHidden, locale, screenLayout, ...
  - Implement onConfigurationChanged() callback (optional):
  - Optional b/c super.onConfigurationChanged() is called
  - Use Configuration$_{object}$ to retrieve updated config
  - Apply updated layouts, etc. from Resources$_{object}$
  - Not recommended for most apps

Service Lifecycle

- Core callbacks:
  - onCreate()
  - onDestroy()
- Ways to operate a Service (not mutually exclusive):
  - Start/stop the service
  - Bind/unbind to the Service


Service is started by startService()
onCreate()
onStart()
Service is running
The service is stopped (no callback)
onDestroy()
Service is shut down
Service is created by bindService()
onCreate()
onBind()
Client interacts with the service
onRebind()
onUnbind()
onDestroy()
Service is shut down

- Start/stop Service:
  - Context.startService(), generates onStart() callback w/ $\text{Intent}_{\text{object}}$
  - Context.stopService(), no "onStop()" callback, just onDestroy()
  - Service.stopSelf()
  - Service.stopSelfResult()
- Bind/unbind to the Service:
  - Context.bindService(), generates onBind()/onRebind callback
  - Context.unbindService(), generates onUnbind() callback
  - Client-Service communication through IDL-defined interface

Broadcast Receiver Lifecycle

- Only has onReceive()
- Considered active only while servicing this call
- Encompassing process "protected" while Broadcast Receiver active
- Should do minimal work and return
- Should start service for long-running work

Content Provider Lifecycle

- Only has onCreate()
- Implement Content Provider REST-like callbacks:
  - query(Uri, String[], String, String[], String)
  - insert(Uri, ContentValues)
  - update(Uri, ContentValues, String, String[])
  - delete(Uri, String, String[])
  - getType(Uri) returns Content Provider data MIME type

Processes and Lifecycles

- Android's process management is predicated on low-memory:
  - Processes are started and remain active as long as RAM is available
  - When RAM is low, system starts "garbage collecting"
  - Processes of lower importance are killed to free up space
  - System continously maintains an active Components "importance hierarchy"
  - System tries to provide "priority inheritance"
- Hierarchy:
  1. Foreground process - killed as a last resort
  2. Visible process
  3. Service process
  4. Background process
  5. Empty process

- Foreground process:
  - Running user-facing Activity (onResume() called)
  - Hosting Service bound to user-facing Activity
  - Hosting Service executing Lifecycle callback
  - Hosting Broadcast Receiver executing onReceive() callback
- Visible process:
  - Running user-visible Activity
  - Hosting Service bound to user-visible Activity
- Service process:
  - Any service that isn't of a higher priority in hierarchy
- Background process - maintained in LRU list:
  - Running non-visible Activity
- Empty process:
  - One not holding any Activity, mainly maintained for cache

User Interface

Opersys inc.

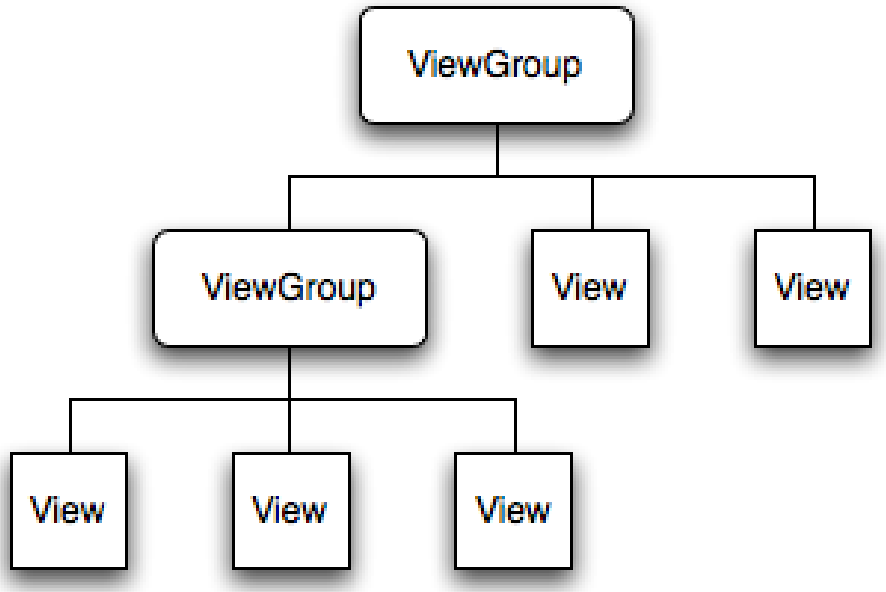
August 12, 2012

Basics

- UI is built on top of $\text{View}_{\text{object}}$ and $\text{ViewGroup}_{\text{object}}$
- $\text{View}_{\text{class}}$ is root for subclassed "widgets" (text fields, buttons, etc.)
- $\text{ViewGroup}_{\text{class}}$ is root for subclassed "layouts"
- $\text{View}_{\text{object}}$ = rectangle on screen. Handled by object for rectangle:
  - Measurement
  - Layout
  - Drawing
  - Focus change
  - Scrolling
  - Key/gesture interactions

View hierarchy

- Activity UI defined as $\text{View}_{\text{object}}$ and $\text{ViewGroup}_{\text{object}}$ hierarchy
- $\text{View}_{\text{object}}$ = leaves
- $\text{ViewGroup}_{\text{object}}$ = branch



- Use setContentView() to make hierarchy visible
- Drawing is done in-order
- Ergo: Last element drawn is on top
- Use $\text{hierarchyviewer}_{\text{tool}}$ to view your app's hierarchy

Layout

- Declare in XML (statically) or in Java (at runtime)
- Easiest to boot: XML
- Coherent element nomenclature:
  - `<TextView>` = $\text{TextView}_{\text{class}}$
  - `<LinearLayout>` = $\text{LinearLayout}_{\text{class}}$
- System creates Java objects corresponding to XML layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

- Nested layout elements allow creating tree
- More layout elements exist:
  - LinearLayout
  - RelativeLayout
  - TableLayout
  - GridLayout
  - ...
- Use addView(View) at runtime to insert additional $\text{View}_{\text{object}}$ and/or $\text{ViewGroup}_{\text{object}}$

Widgets

- Widget = $\text{View}_{\text{object}}$ w/ specific UI:
  - Buttons
  - Checkboxes
  - Text-entry field
  - Date picker
  - Clock
  - Zoom controls
- See $\text{android.widget}_{\text{package}}$
- You can create custom widgets

UI events

- To get events:
  - Define and register event listener, or
  - Override existing one of widget's callbacks
- Event listener:
  - Most common case
  - $\text{View}_{\text{class}}$ contains collections of nested interfaces w/ callbacks
  - Must implement interface/callback and register it to $\text{View}_{\text{object}}$
  - Generic form:
    - $\text{On*Listener}_{\text{interface}}$ & On*() callback; View.setOn*Listener()

- Event listener (cntd):
  - Examples:
    - View.OnClickListener & onClick(); View.setOnClickListener()
    - View.OnTouchListener & onTouch(); View.setOnTouchListener()
    - View.OnKeyListener & onKey(); View.setOnKeyListener()
- Callback override:
  - For custom widgets
  - Actual $\text{Widget}_{\text{object}}$ callbacks:
    - onTouchEvent()
    - onKeyDown()
    - onKeyUp()

Menus

- Types:
  - Main app menu view through the MENU key
  - Contextual menus
- Menus are $\text{View}_{\text{object}}$ hierarchies too
- However:
  - Hierarchy is automatically created by system, not you
  - No need to register event listeners
- Instead, implement callbacks for:
  - Populating menu:
    - onCreateOptionsMenu()
    - onCreateContextMenu()
  - Handling menu selection:
    - onOptionsItemSelected()
    - onContextItemSelected()

Advanced topics

- Adapters
  - For displaying variable lists of data (ex: tweet feed)
  - $\text{AdapterView}_{\text{class}}$ is subclass of $\text{ViewGroup}_{\text{class}}$
  - $\text{View}_{\text{object}}$ children populated w/ data from object implementing $\text{Adapter}_{\text{interface}}$
  - Example $\text{Adapter}_{\text{object}}$:
    - CursorAdapter, for DB cursor reading
    - ArrayAdapter, for array reading
- Styles and themes
  - Widget theming
  - Style:
    - Formatting attributes applied to individual/specific elements
    - Ex: Text size and color
  - Theme:
    - Formatting attributes applied to an Activity or many Activities
    - Ex: Window color, panel bg, text sizes, colors
  - Style and themes are resources
  - Default styles and themes included

Declaring layout

- Declare in either or both:
  - XML (statically)
  - Java (at runtime / programmatically)
- Ex: Declare statically & modify dynamically
- XML benefit: Separate presentation from code
- Ex: Let designer take care of XML and programmer of code

Write the XML

- Create a 'res/layout/*.xml' file with XML layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

Load the XML resource

- On compile, a $\text{View}_{\text{resource}}$ is created based on XML
- Loading resource at Activity creation:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView.(R.layout.main_layout);
}
```

- The resource is R.layout.* where *.xml is the filename in 'res/layout'

Attributes

- Types:
  - Inherited from 'View' root class (ex: $\text{id}_{\text{attribute}}$)
  - Specific to subclassed widget (ex: $\text{textSize}_{\text{attribute}}$ of $\text{TextView}_{\text{class}}$)
  - Layout parameters applied to all $\text{View}_{\text{objects}}$ of parent $\text{ViewGroup}_{\text{object}}$

ID

- $\text{View}_{\text{objects}}$ can have unique IDs in hierarchy tree:
  - Specified as string in XML
  - Compiled as unique integer
- '@+' tells $\text{aapt}_{\text{tool}}$ to create new resource integer

```
android:id="@+id/my_button"
```

- Can then refer to $\text{View}_{\text{object}}$ in code:

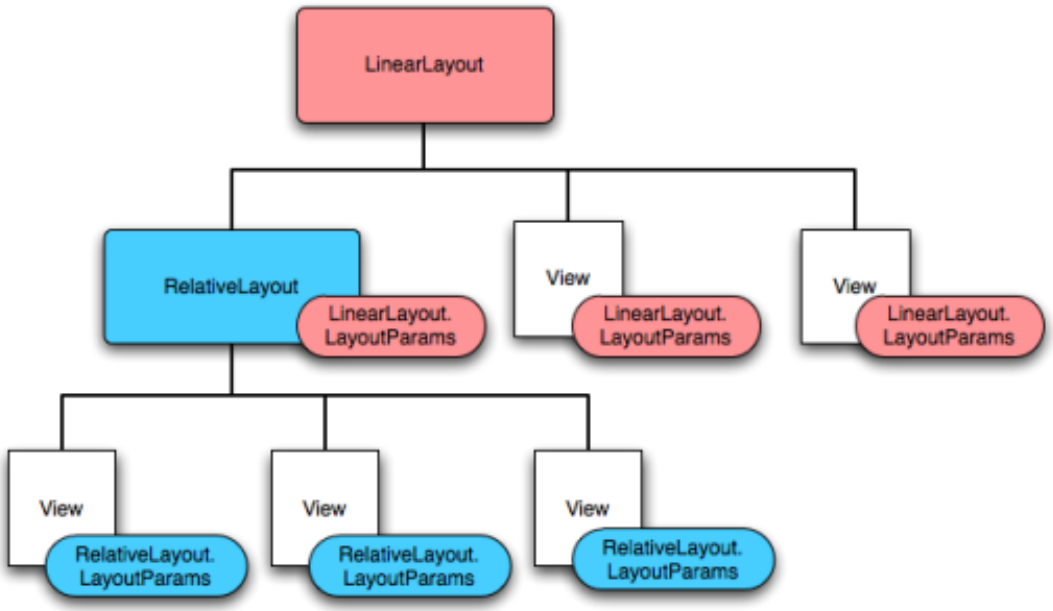
```
Button myButton = (Button) findViewById(R.id.my_button);
```

- Android resource IDs can be referenced without '+':

```
android:id="@android:id/empty"
```

Layout parameters

- Each layout type has layout_* Attributes
- Common to all:
  - layout_width
  - layout_height
- Layout parameters apply to ViewGroup$_{object}$'s children



- Values can be absolute (not recommended) or relative (recommended)
- Common presets for width—height:
  - wrap_content
  - fill_parent (match_parent since API level 8)
- More on values in resource discussion

Layout position

- $\text{View}_{\text{object}}$ is rectangle
- Can get $\text{View}_{\text{object}}$'s position in px programmatically:
  - getLeft()
  - getTop()
  - getWidth()
  - getRight() = getLeft() + getWidth()

Size, padding and margins

- Two sizes:
  - Desired width&height - a.k.a. "measured" width&height
    - getMeasuredWidth()
    - getMeasuredHeight()
  - Actual width&height - a.k.a. "drawing" width&height
    - getWidth()
    - getHeight()
- "drawing" values are those **after rendering**
- View&Object padding:
  - setPadding(int, int, int, int)
  - getPaddingLeft()
  - getPaddingTop()
  - getPaddingRight()
  - getPaddingBottom()
- Margins defined in $\text{ViewGroup}_{\text{object}}$: see $\text{ViewGroup.MarginLayoutParams}_{\text{class}}$

Creating menus

- Types:
  - $\text{Options}_{\text{menu}}$ - MENU key:
    - Icon menu, 6 items max
    - Expanded menu, "More" menu
  - $\text{Context}_{\text{menu}}$ - long-press on $\text{View}_{\text{object}}$
  - $\text{Submenu}_{\text{menu}}$ - in $\text{Options}_{\text{menu}}$—$\text{Context}_{\text{menu}}$
    - No nested submenus

Defining menus

- Same as layouts
- Use 'res/menu/*.xml' file to define $*_{menu}$
- Elements:
  - <menu> = $Menu_{object}$, root element
  - <item> = $MenuItem_{object}$
  - <group> for grouping <item> elements
- Submenus:

```
<menu>
  <item>
     <menu>
     ...
     </menu>
  </item>
</menu>
```

- Example 'res/menu/game_menu.xml':

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/new_game"
          android:icon="@drawable/ic_new_game"
          android:title="@string/new_game" />
    <item android:id="@+id/quit"
          android:icon="@drawable/ic_quit"
          android:title="@string/quit" />
</menu>
```

Inflating a menu resource

- Convert XML to programmable object:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.game_menu, menu);
    return true;
}
```

- Activity's onCreateOptionsMenu() called on 1st MENU key press
- Can also use Menu.add() to add items

Creating an options menu

- On click triggers onOptionsItemSelected() callback
- Use switch-case on item's ID to perform action
- Return 'true' = handled successfully

- Example:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
    case R.id.new_game:
        newGame();
        return true;
    case R.id.quit:
        quit();
        return true;
    default:
        return super.onOptionsItemSelected(item);
    }
}
```

Share menu

- If multiple activities share same $\text{Options}_{\text{menu}}$:
  - Create Activity superclass containing menu
  - Sublcass all Activities from superclass
- To extend menu in subclass:
  - Override onCreateOptionsMenu() in subclass
  - Call super.onCreateOptionsMenu()
  - Use Menu.add() in subclass

Changing the menu when it opens

- onCreateOptionsMenu() called only once
- onPrepareOptionsMenu() called w/ current menu
- Override onPrepareOptionsMenu() to modify menu (disable, add, remove items)
- Do **NOT** use to implement per in-focus $\text{View}_{\text{object}}$:
  - No focus in case of touch-triggered $\text{View}_{\text{object}}$ elements
  - Use context menu instead

Creating a context menu

- "Right-click" equivalent
- Activated on long-press on $\text{View}_{\text{object}}$
- No icons or shortcuts
- Most often implemented for items in $\text{ListView}_{\text{object}}$
- To enable: registerForContextMenu() and pass $\text{View}_{\text{object}}$
- For ListView:

```
registerForContextMenu(getListView());
```

- To inflate: override onCreateContextMenu()
- To handle: override onContextItemSelected()

- Ex inflating:

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenuInfo menuInfo) {
  super.onCreateContextMenu(menu, v, menuInfo);
  MenuInflater inflater = getMenuInflater();
  inflater.inflate(R.menu.context_menu, menu);
}
```

- Ex handling:

```
@Override
public boolean onContextItemSelected(MenuItem item) {
  AdapterContextMenuInfo info = (AdapterContextMenuInfo) item.getMenuInfo();
  switch (item.getItemId()) {
  case R.id.edit:
    editNote(info.id);
    return true;
  case R.id.delete:
    deleteNote(info.id);
    return true;
  default:
    return super.onContextItemSelected(item);
  }
}
```

Creating submenus

- Use <menu> element inside <item> element in menu XML file
- Handle in onOptionsItemSelected() switch-case
- Can be created programmatically:
  - Menu.addSubMenu() returns $\text{SubMenu}_{\text{object}}$
  - SubMenu.add() to add items

Other menu features

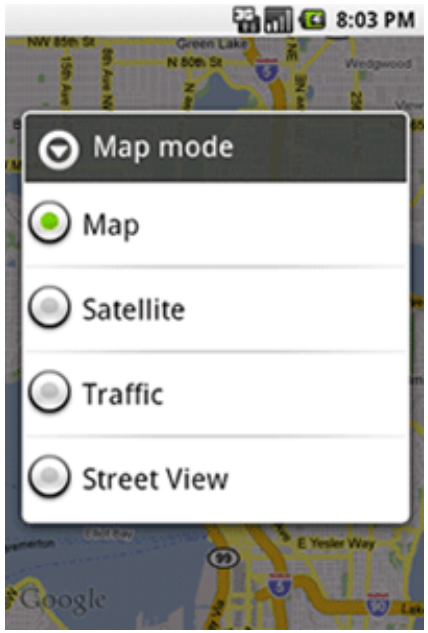
- Menu groups
- Checkable menu items
- Shortcut keys
- Intents for menu items

Menu groups

- Created by nesting `<item>` elements inside `<group>`
- Shared traits:
  - setGroupVisible() - show—hide items
  - setGroupEnabled() - enable—disable items
  - setGroupCheckable() - are all items checkable?

Checkable menu items

- Mostly for context menus
- Requires manual handling for icon switching in options menu
- Use android:checkableBehavior$_{\text{attribute}}$ in <group> element:
  - 'single' - radio buttons
  - 'all' - checkboxes
  - none



- Example:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:checkableBehavior="single">
        <item android:id="@+id/red"
              android:title="@string/red" />
        <item android:id="@+id/blue"
              android:title="@string/blue" />
    </group>
</menu>
```

- Per-item:
  - android:checked$_{\text{attribute}}$
  - setChecked()
- Item 'checking' is manual, not automatic
- Use isChecked() and setChecked() in onOptionsItemSeleted()
- State preserved during Activity lifetime
- Use Shared Preferences to save/restore state

- Example isChecked()/setChecked() use:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
  switch (item.getItemId()) {
  case R.id.vibrate:
  case R.id.dont_vibrate:
    if (item.isChecked()) item.setChecked(false);
    else item.setChecked(true);
    return true;
  default:
    return super.onOptionsItemSelected(item);
  }
}
```

Shortcut keys

- Available only when device has hardware keyboard
- Not case sensitive
- In `<item>` use:
  - android:alphabeticShortcut$_{attribute}$
  - android:numericShortcut$_{attribute}$
- Programmatically:
  - setAlphabeticShortcut(char)
  - setNumbericShortcut(char)
- Ex: ("Menu is open" — "MENU key presssed") & "s" = "save"
- Shortcut displayed as "tip" below menu item name in menu

Intents for menu items

- Start Activity from menu
  - Call startActivity() w/ appropriate Intent in onOptionsItemSelected()
  - Inoperative menu item if no Activity handles Intent on device
- System can automagically add menu items if Intent-handling Activity exists
  - Based on using menu.addIntentOptions() in onCreateOptionsMenu()
  - OOS: See "Dynamically adding intents"
- Can allow your Activity to be added to others' menus
  - Based on using $\text{ALTERNATIVE}_{\text{category}}$ and $\text{SELECTED\_ALTERNATIVE}_{\text{category}}$ in Manifest `<intent-filter>`
  - OOS: See "Allowing your Activity to be added to menus"

Creating dialogs

- Small window appearing in front of Activity
- Types:
  - AlertDialog, can be used to construct most dialogs
  - ProgressDialog, progress wheel or bar
  - DatePickerDialog
  - TimePickerDialog
- Typically created programmatically in Java
- Can create custom dialogs using XML

Showing a dialog

- Always part of Activity
- To display:
  - Define unique IDs for each dialog
  - Call showDialog() w/ ID when appropriate
  - Implement switch-case in onCreateDialog() to create dialogs
  - Implement switch-case in onPrepareDialog() to modify after creaion

- Example dialog creation:

```
protected Dialog onCreateDialog(int id) {
    Dialog dialog;
    switch(id) {
    case DIALOG_PAUSED_ID:
        // do the work to define the pause Dialog
        break;
    case DIALOG_GAMEOVER_ID:
        // do the work to define the game over Dialog
        break;
    default:
        dialog = null;
    }
    return dialog;
}
```

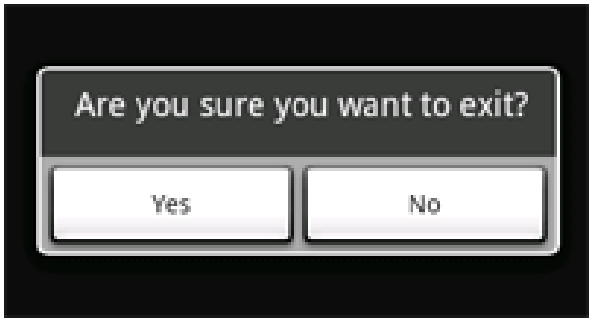
Dismissing a dialog

- Dimissing dialog:
  - Dialog.dismiss()
  - dismissDialog(int)
- Dimissing dialog (if shown) and removing all references:
  - removeDialog(int)
- Listening to dismissal:
  - DialogInterface.OnDismissListener$_{\text{interface}}$ ->onDismiss(DialogInterface)
  - Use Dialog.setOnDismissListener()
- Listening for cancel (BACK key or "Cancel" selected):
  - DialogInterface.OnCancelListener$_{\text{interface}}$ ->onCancel(DialogInterface)
  - Use Dialog.setOnCancelListener()

Creating an AlertDialog

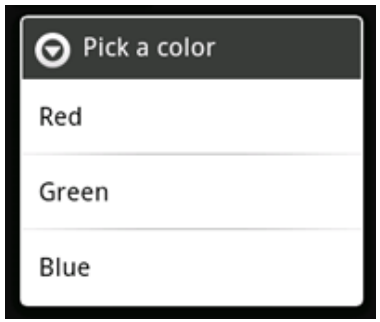
- Use for dialogs containing any of:
  - Title
  - Text message
  - One, two, or three buttons
  - Selectable items list (checkboxes, radio buttons)
- Instantiate $\text{AlertDialog.Builder}_{\text{object}}$ using current $\text{Context}_{\text{object}}$
- Call on AlertDialog.Builder.* to populate dialog
- Retrieve $\text{AlertDialog}_{\text{object}}$ using AlertDialog.Builder.create()

Adding buttons



```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setMessage("Are you sure you want to exit?")
       .setCancelable(false)
       .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
           public void onClick(DialogInterface dialog, int id) {
                MyActivity.this.finish();
           }
       })
       .setNegativeButton("No", new DialogInterface.OnClickListener() {
           public void onClick(DialogInterface dialog, int id) {
                dialog.cancel();
           }
       });
AlertDialog alert = builder.create();
```

Adding a list

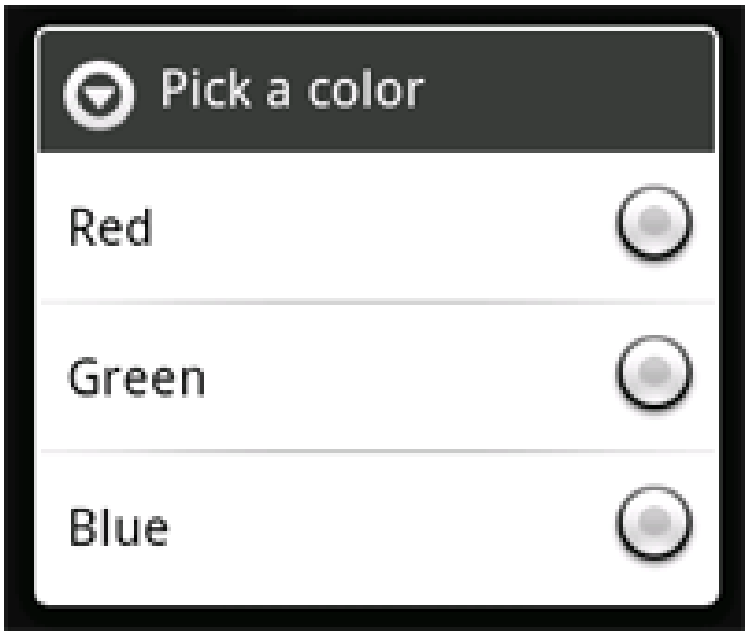



```
final CharSequence[] items = {"Red", "Green", "Blue"};

AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setTitle("Pick a color");
builder.setItems(items, new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int item) {
        Toast.makeText(getApplicationContext(), items[item], Toast.LENGTH_SHORT).sh
    }
});
AlertDialog alert = builder.create();
```

Adding checkboxes and radio buttons

- Checkboxes: setMultiChoiceItems()
- Radio buttons: setSingleChoiceItems()
- Choices preservered during Activity lifetime
- Use Activity Lifecycle events and/or permanent storage to safeguard

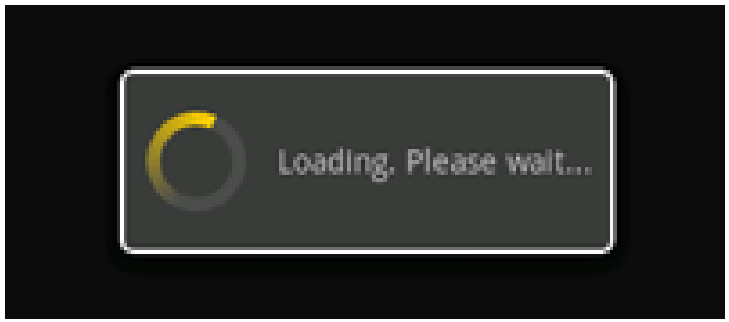


```
final CharSequence[] items = {"Red", "Green", "Blue"};

AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setTitle("Pick a color");
builder.setSingleChoiceItems(items, -1, new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int item) {
        Toast.makeText(getApplicationContext(), items[item], Toast.LENGTH_SHORT).sh
    }
});
AlertDialog alert = builder.create();
```

Creating a ProgressDialog

- ProgressDialog is AlertDialog subclass
- No need to a $\text{Builder}_{\text{object}}$
- Use ProgressDialog.show() directly
- Can use dismiss() when relevant




```
ProgressDialog dialog = ProgressDialog.show(MyActivity.this, "",
                        "Loading. Please wait...", true);
```

Showing a progress bar

- Steps:
  1. Pass $\text{Context}_{\text{object}}$ to $\text{ProgressDialog}_{\text{class}}$ constructor
  2. Use setProgressStyle() to set style to "STYLE_HORIZONTAL"
  3. Set all other dialog properties
  4. Show dialog with show() or return $\text{ProgressDialog}_{\text{object}}$ from onCreateDialog(int)
  5. Use setProgress(int) or incrementProgressBy(int) to show progress
- Example setup:

```
ProgressDialog progressDialog;
progressDialog = new ProgressDialog(mContext);
progressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
progressDialog.setMessage("Loading...");
progressDialog.setCancelable(false);
```

- Use a $\text{Handler}_{\text{class}}$ thread and:
  - Start separate thread for handling operation
  - Send messages from thread to report progress
  - Receive messages from thread and modify progress
  - Dismiss dialog and thread when progress = 100%

Other dialogs

- For DatePickerDialog, OOS: see "Hello DataPicker" tutorial
- For TimePickerDialog, OOS: see "Hello TimePicker" tutorial

Creating a custom dialog

- Use 'res/layout/*.xml' file to define custom XML dialog
- Use dialog.setContentView(int) to set dialog's content
- OOS: See "Creating a Custom Dialog"

Handling UI events

- Two ways:
  - Event listeners, most common way
  - $\text{View}_{\text{object}}$ callbacks, typically for custom widgets
    - Requires subclassing widget (heavy for common use)

Event listeners

- $\text{View}_{\text{class}}$ interface containing single callback:
  - $\text{View.OnClickListener}_{\text{interface}}$ & onClick()
  - $\text{View.OnLongClickListener}_{\text{interface}}$ & onLongClick()
  - $\text{View.OnFocusChangeListener}_{\text{interface}}$ & onFocusChange()
  - $\text{View.OnKeyListener}_{\text{interface}}$ & onKey()
  - $\text{View.OnTouchLitener}_{\text{interface}}$ & onTouch()
  - View.OnCreateContextMenuListener & onCreateContextMenu()
- Implement listener and call View.set*Listener()

- Example listener within Activity:

```
// Create an anonymous implementation of OnClickListener
private OnClickListener mCorkyListener = new OnClickListener() {
    public void onClick(View v) {
      // do something when the button is clicked
    }
};

protected void onCreate(Bundle savedValues) {
    ...
    // Capture our button from layout
    Button button = (Button)findViewById(R.id.corky);
    // Register the onClick listener with the implementation above
    button.setOnClickListener(mCorkyListener);
    ...
```

- Example listener Actvity implement listener interface:

```
public class ExampleActivity extends Activity implements OnClickListener {
    protected void onCreate(Bundle savedValues) {
        ...
        Button button = (Button)findViewById(R.id.corky);
        button.setOnClickListener(this);
    }

    // Implement the OnClickListener callback
    public void onClick(View v) {
      // do something when the button is clicked
    }
    ...
}
```

- Some callbacks have boolean return value:
  - onLongClick()
  - onKey()
  - onTouch()
- 'true' = event handled + don't propagate
- 'false' = event not handled + propagate
- For onTouch(), 'false' also means no interest if further touch events

Event handlers

- $\text{View}_{\text{object}}$ callbacks called when user interacts with:
  - Touch screen
  - Keyboard
  - Trackball
- OOS: See "Event Handlers" and "Building Custom Components"

Touch mode

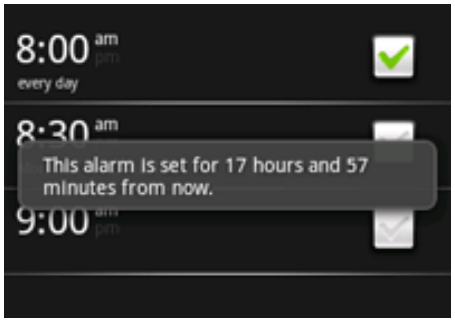
- Device is either in or out of touch mode
- Query global isInTouchMode() for current state
- Entry triggered when user touches screen
- In touch mode:
  - Only some widgets are focusable (ex: text entry)
- Out of touch mode:
  - Directional key and trackball permit focus select

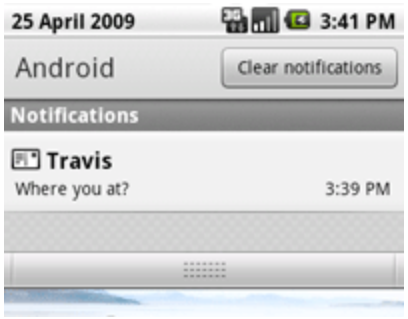
Handling focus

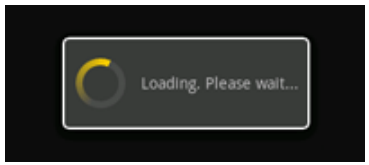
- System automagically takes care of focus (fg, bg, new View, etc.)
- Can override default system bevior
- OOS: See "Handling Focus"

Notifying the user

- Types:
  - Toast notification
  - Status Bar notification
  - Dialog notification







Creating toast notifications

- Created from Activity or Service
- Does not allow user to respond

The basics

- Use $\text{Toast}_{\text{object}}$'s makeText()
- Very simple use
- Example - the long version:

```
Context context = getApplicationContext();
CharSequence text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(context, text, duration);
toast.show();
```

- Example - the short version:

```
Toast.makeText(getApplicationContext(), "Hello toast!", Toast.LENGTH_SHORT).sh
```

Positioning your toast

- Std location: centered, near-bottom
- Use setGravity(Gravity$_{\text{class}}$ constant, x-position, y-position) to change
- Example:

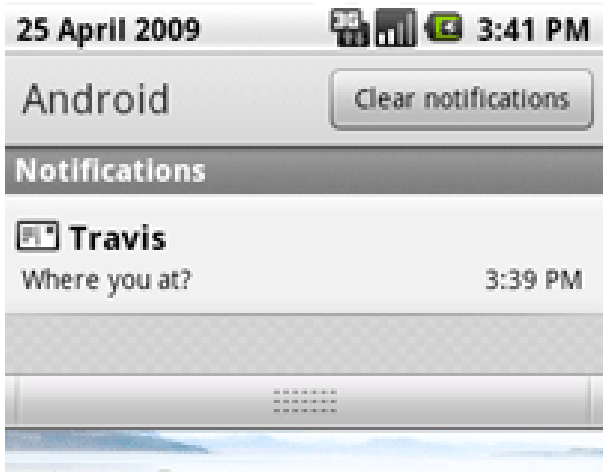
```
toast.setGravity(Gravity.TOP|Gravity.LEFT, 0, 0);
```

Creating a custom toast View

- Use 'res/layout/*.xml' file to define toast layout
- Inflate layout using $\text{LayoutInflater}_{\text{object}}$
- Use $\text{Toast}_{\text{object}}$'s setView() to toast's layout
- Show toast using show()
- OOS: See "Creating a Custom ToastView"

Creating status bar notifications

- Adds:
  - Icon on system status bar
  - Expanded message in Notifications$_{window}$
- Selection of expanded message triggers notification-defined Intent
- Said Intent is typically tied to pre-defined Activity
- Status bar notifications mainly used by Service to notify user
  - Service should never start Activity directly



The basics

- Initiated by Activity or Service
- Typically from Service
- Use:
  - $\text{Notification}_{\text{class}}$, defines:
    - Icon
    - Expanded message
    - Extra settings (sound, vibration, etc.)
  - $\text{NotificationManager}_{\text{class}}$:
    - System service managing and servicing all notifications
    - Retrieved using getSystemService(NOTIFICATION_SERVICE)
  - Pass $\text{Notification}_{\text{object}}$ to retrieved $\text{NotificationManager}_{\text{object}}$'s notify()
- Example:

1. Get a NotificationManager$_{object}$ reference:

```
String ns = Context.NOTIFICATION_SERVICE;
NotificationManager mNotificationManager = (NotificationManager) getSystemServ
```

2. Instantiate Notification$_{object}$:

```
int icon = R.drawable.notification_icon;
CharSequence tickerText = "Hello";
long when = System.currentTimeMillis();

Notification notification = new Notification(icon, tickerText, when);
```

3. Define Notification$_{object}$'s expanded message and Intent:

```
Context context = getApplicationContext();
CharSequence contentTitle = "My notification";
CharSequence contentText = "Hello World!";
Intent notificationIntent = new Intent(this, MyClass.class);
PendingIntent contentIntent = PendingIntent.getActivity(this, 0, notificationI

notification.setLatestEventInfo(context, contentTitle, contentText, contentInt
```

4. Pass the Notification$_{object}$ to the NotificationManager$_{object}$:

```
private static final int HELLO_ID = 1;

mNotificationManager.notify(HELLO_ID, notification);
```

Managing your notifications

- 1st param to NotificationManager$_{\text{object}}$'s notify() is app-unique ID
- Can use ID to:
  - Update notification
  - Identify notification when Intent is received by Activity
  - Cancel notification: NotificationManager.cancel(int)
- More on notification cancelling:
  - FLAG_AUTO_CANCEL$_{\text{flag}}$ in Notification$_{\text{object}}$ ensures auto-cancelling after select
  - NotificationManager.cancelAll()

Creating a notification

- Required:
  - Icon for status bar
  - Title and expanded message for expanded view
  - PendingIntent$_{\text{object}}$ for firing on select
- Optional:
  - Ticker-text for status bar
  - Alert sound
  - Vibrate setting
  - Flashing LED setting

Update the notification

- Preferable to cluttering expanded view
- Steps:
  1. Update notification values
  2. Call Notification.setLatestEventInfo()
  3. Call NotificationManager.notify()

Adding a sound

- Default sound (overrides $\text{sound}_{\text{field}}$ if set):

```
notification.defaults |= Notification.DEFAULT_SOUND;
```

- Custom sound from file:

```
notification.sound = Uri.parse("file:///sdcard/notification/ringer.mp3");
```

- Custom sound from Content Provider:

```
notification.sound = Uri.withAppendedPath(Audio.Media.INTERNAL_CONTENT_URI, "6
```

Adding vibration

- Default pattern (overrides $\text{vibrate}_{\text{field}}$ if set):

```
notification.defaults |= Notification.DEFAULT_VIBRATE;
```

- Custom vibration
- Pass array of:
  1. Start
  2. Length of 1st vibration
  3. Length of 2nd vibration
     - ...
     - n+1. Length of n'th vibration
- No loop possible

```
long[] vibrate = {0,100,200,300};
notification.vibrate = vibrate;
```

Adding flashing lights

- Default setting:

```
notification.defaults |= Notification.DEFAULT_LIGHTS;
```

- Custom pattern:
  - Define:
    - ledARGB: color
    - ledOffMS: milli-seconds to keep light off
    - ledOnMS: milli-seconds to keep light on
  - Hardware support differs, system will do best effort
  - Green is most common

```
notification.ledARGB = 0xff00ff00;
notification.ledOnMS = 300;
notification.ledOffMS = 1000;
notification.flags |= Notification.FLAG_SHOW_LIGHTS;
```

More features

- $\text{FLAG\_AUTO\_CANCEL}_{\text{flag}}$: Auto-cancel on select
- $\text{FLAG\_INSISTENT}_{\text{flag}}$: Repeat audio until user action
- $\text{FLAG\_ONGOING\_EVENT}_{\text{flag}}$: Notification shown under "Ongoing" in expanded view
- $\text{FLAG\_NO\_CLEAR}_{\text{flag}}$: Do not clear if "Clear notifications" button selected
- $\text{number}_{\text{field}}$: Nbr of events represented (must start with "1")
- $\text{iconLevel}_{\text{field}}$: Useful for animated icon (see $\text{LevelListDrawable}_{\text{class}}$)
- See $\text{Notification}_{\text{class}}$ doc for more features

Creating a custom expanded view

- General steps:
  1. Use 'res/layout/*.xml' file to define custom expanded view
  2. Use $\text{RemoteViews}_{\text{object}}$ to define image and text
  3. Set $\text{Notification}_{\text{object}}$'s $\text{contentIntent}_{\text{field}}$
  4. Call NotificationManager.notify()
- Can be used to add $\text{Chronometer}_{\text{object}}$ or $\text{ProgressBar}_{\text{object}}$
- OOS: See "Creating a Custom Expanded View"

Applying styles and themes

- Style:
  - Collection of properties specifying $\text{View}_{\text{object}}$ or window look and format
  - Defined in XML resource seperate from layout XML
  - Useful for "lightening" layout XML
  - Ex:
    - Replace:

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textColor="#00FF00"
    android:typeface="monospace"
    android:text="@string/hello" />
```

    - With:

```
<TextView
    style="@style/CodeFont"
    android:text="@string/hello" />
```

- Theme:
  - Style applied to entire Activty or app, not just individual $\text{View}_{\text{object}}$
- OOS: See "Applying Styles and Themes"

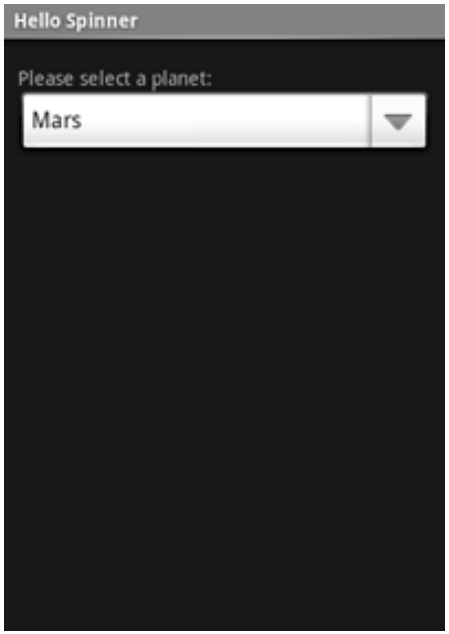
Building custom components

- Android provides a slew of View- and ViewGroup-based widgets
- If none meet your needs, you can define your own
- Use-case examples:
  - Custom-rendered View, like a Custom 2D analog control knob
  - Combine View widgets into one, like a ComboBox
  - Override default widgets rendering, like EditText in Notepad example
  - Custom-handling of events, like for a game
- OOS: See "Building Custom Components"

Binding data with AdapterView

- Useful when displaying stored data
- $\text{ViewGroup}_{\text{class}}$ subclass w/ $\text{Adapter}_{\text{object}}$-filled $\text{View}_{\text{object}}$ children
- Responsibilities:
  - Filling layout with data
  - Handling user selections
- Examples:
  - Gallery
  - ListView (very useful, very common)
  - Spinner

The "Hello Views" - Spinner example



1. Create a new project
2. Insert into 'res/layout/main.xml'

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:padding="10dip"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dip"
        android:text="@string/planet_prompt"
    />
    <Spinner
        android:id="@+id/spinner"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:prompt="@string/planet_prompt"
    />
</LinearLayout>
```

3. Create 'res/values/strings.xml'

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="planet_prompt">Choose a planet</string>
    <string-array name="planets_array">
        <item>Mercury</item>
        <item>Venus</item>
        <item>Earth</item>
        <item>Mars</item>
        <item>Jupiter</item>
        <item>Saturn</item>
        <item>Uranus</item>
        <item>Neptune</item>
    </string-array>
</resources>
```

4. In 'HelloSpinner.java':

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    Spinner spinner = (Spinner) findViewById(R.id.spinner);
    ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(
            this, R.array.planets_array, android.R.layout.simple_spinner_item);
    adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
    spinner.setAdapter(adapter);
}
```

- createFromResource() binds each array item to initial $\text{Spinner}_{\text{object}}$ appearance
- android.R.layout.simple_spinner_item is a layout for std Spinner appearance
- setDropDownViewResource() defines appearance of each item when widget is open
- android.R.layout.simple_spinner_dropdown_item is another layout
- Reminder: android.R.* are platform-defined resources

5. Implement class that provides callback for seleted item:

```
public class MyOnItemSelectedListener implements OnItemSelectedListener {

    public void onItemSelected(AdapterView<?> parent,
        View view, int pos, long id) {
      Toast.makeText(parent.getContext(), "The planet is " +
          parent.getItemAtPosition(pos).toString(), Toast.LENGTH_LONG).show();
    }

    public void onNothingSelected(AdapterView parent) {
      // Do nothing.
    }
}
```

6. Go back to onCreate() and add this at the end:

```
spinner.setOnItemSelectedListener(new MyOnItemSelectedListener());
```

7. Run app.

Common layout objects

- Used to visually organize $\text{View}_{\text{object}}$ elements
- Can be nested
- See "Hello Views" tutorial for screenshots
  http://developer.android.com/resources/tutorials/views/index.html

Important ViewGroups

ViewGroup	Description
FrameLayout	Frame to display a single object
Gallery	Horizontally scrolling dispaly of images, from bound list
GridView	Scrolling MxN grid
LinearLayout	Scrollable vertical or horizontal single-row layout
ListView	List of scrollable items
RelativeLayout	Position children relative to each other or parent
ScrollView	Vertically scrollable column of elements
Spinner	Displays single item from bound list
SurfaceView	Provides direct access to drawing surface, typically for pixel-drawing apps
TabHost	Provides tabs and callbacks for changing content
TableLayout	HTML-like table w/ rows and columns
ViewFlipper	Slideshow-like one-row textbox
ViewSwitcher	Same as ViewFlipper
WebView	A web page in a View - Can be used to create RIAs in HTML/CSS/JS

LinearLayout

- Aligns all children either vertically or horizontally
- Provides margins
- Respects gravity (i.e. alignment: right, left, center)
- Support weight (importance of child relative to others)

TableLayout

- Main element: `<TableLayout>`
- Contains: `<TableRow>`
- Each `<TableRow>` child is a column

RelativeLayout

- Allow positioning relative to parent or other children
- Elements rendered in order
- Example XML attributes:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:background="@drawable/blue"
                android:padding="10px" >
    . . .
    <Button android:id="@+id/ok"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_below="@id/entry"
            android:layout_alignParentRight="true"
            android:layout_marginLeft="10px"
            android:text="OK" />

    <Button android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_toLeftOf="@id/ok"
            android:layout_alignTop="@id/ok"
            android:text="Cancel" />
</RelativeLayout>
```

How Android draws Views

OOS: See "How Android Draws Views"

Application Resources

Opersys inc.

August 12, 2012

Introduction

- Goal:
  - Externalize resources for maintaining independtly from source code
  - Support multiple device configurations (language, screen size, etc.)
- Types:
  - Default resource: applies to all configurations
  - Alternative resource: applies to specific configuration
- Location: 'res/*'
- Predefined hierarchy nomenclature w/ automagic selection:
  - Predicated on the use of "-" character to separate qualifiers
  - Ex: 'res/layout-land/' contains layouts applying to "landscape" configuration
  - Can include multiple qualifiers
  - Ex: 'res/layout-en-rUS-land/' = layouts applying to US-english in "landscape"

- Accessed through app's R_{class}
- Vs. "assets":
  - Location: 'assets/*'
  - Accessed by passing full filenames using $AssetManager_{object}$
- Example:

```
MyProject/
    src/
        MyActivity.java
    res/
        drawable/
            icon.png
        layout/
            main.xml
            info.xml
        values/
            strings.xml
```

Providing resources

- Grouping resource types
- Providing alternative resources
- Providing the best device compatibility w/ resources
- How Android finds the best-matching resource
- Known issues

Grouping resource types

- **Default** resource types in 'res/':

'dir/'	description
anim/	XML files defining tween animations
color/	XML files defining state list of colors
drawable/	Bitmap files or specially-formatted XML files
layout/	XML files defining interface layouts
menu/	XML files defining app menus
raw/	Arbitrary file in raw form
values/	XML file containing values such as: arrays.xml, colors.xml, dimens.xml, strings.xml, styles.xml
xml/	Arbitrary XML files to be ready by Resources.getXML()

- Access file '`res/<type>/*`' through `R.<type>.*`

Providing alternative resources

- Provide alternatives for different:
  - Screen sizes
  - Screen densities
  - Keyboard configurations
  - Locales
  - etc.
- Specifying alternatives:
  1. Create new in 'res/' matching: `<type>-<qualifier(s)>`
  2. Save alternative resource in new dir using same filename
- Android will automagically select resource matching config
- Example:

```
res/
    drawable/
        icon.png
        background.png
    drawable-hdpi/
        icon.png
        background.png
```

- Valid qualifiers, in order of precedence:

Qualifier	Examples
MCC and MNC	mcc310, mcc310-mnc004, mcc208-mnc00
Lang an region	en, fr, en-rUS, fr-rFR, fr-rCA
Screen size	small, normal, large, xlarge (tablet)
Screen aspect	long, notlong (aspect ratio)
Screen orientation	port, land
Dock mode	car, desk
Night mode	night, notnight
Screen desnity	ldpi, mdpi, hdpi, xhdpi, nodpi
Touchscreen type	notouch, stylus, finger
Keyboard availability	keysexposed, keyssoft
Primary text input	nokeys, qwerty, 12key
Navigation key availability	navexposed, navhidden
Non-touch navigation	nonav, dpad, trackball, wheel
Sys version/API	v3, v4, v7, v9

Qualifier name rules

- Can use "-" to specify multiple qualifier **types**
- Qualifiers must be in order, see above
- Nesting not permitted
- Values are case-insensitive
- Only one value per qualifier type, use aliases if needed

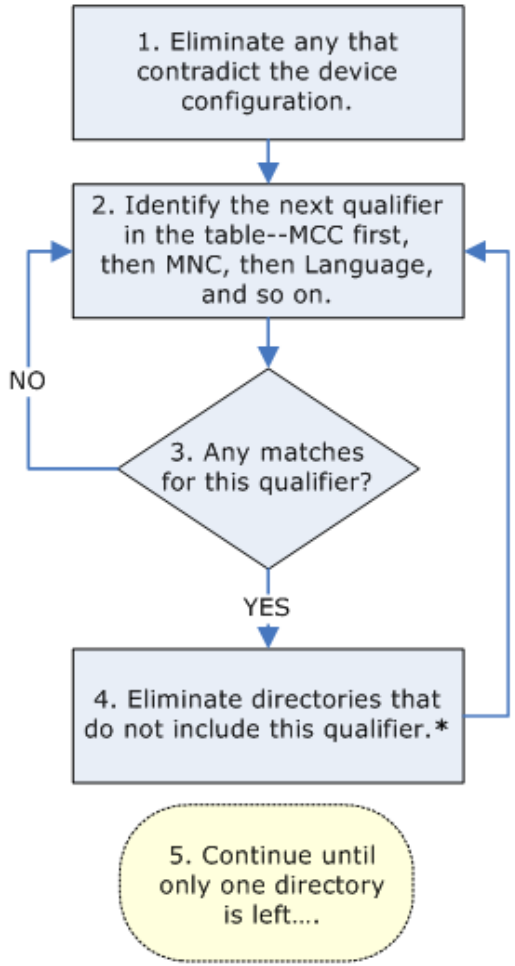
Creating alias resources

- Link qualified resources to alternative in default dir
- Example:
  - "icon.xml" in 'drawable-fr-rCA/' and 'drawable-en-rCA/' points to drawable/icon_ca.png
- OOS: See "Creating alias resources"

Providing the best device compatibility w/ resources

- **Always** provide default resources
  - Even if you have qualified resources
- Add qualified resources as needed
- New Android versions add new qualifiers
- OOS: See "Providing screen resource compatibility for Android 1.5"

How Android finds the best-matching resource



* If the qualifier is the screen density, Android selects a "best" match and the process is done.

Known issues

- Versions prior to 2.0.1 had some issues w/ version qualifier
- OOS: See "Known Issues"

Accessing resources

- $\text{aapt}_{\text{tool}}$ automatically generates R_{class}
- There is `R.<type>` subclass for each '`res/<type>`' dir
- There is `R.<type>.<name>` static integer ID
- `<name>` is either:
  - Filename without extension, or
  - Value in XML if resource is simple value (ex: string)

Access in code

- Typical: `R.<type>.<name>`
- Full: `[<package_name>.]R.<type>.<name>`
- Example:
  - 'res/drawable/myimage.png' = R.drawable.myimage
  - "hello_message" string in 'res/values/strings.xml' = R.string.hello_message

Access in XML

- Typical: `@<type>/<name>`
- Full: `@[<package_name>:]<type>/<name>`
- Example:
  - In layout: `<EditText ... android:text="@string/hello" />`
  - As alias: `<bitmap ... android:src="@drawable/other_drawable" />`

Referencing style attributes

- Reference to value of attribute in current theme
- Full: `?[<package_name>:][<type>/]<name>`
- Ex: `<EditText ... android:textColor="?android:textColorSecondary" ... />`
- Type is omitted/implicit ("attr"): `?android:attr/textColorSecondary`

Accessing platform resources

- Android contains a nbr of std resources (styles, themes and layouts)
- Prepend resource reference with "android" package name to access
- Ex:

```
setListAdapter(new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, myarray));
```

Handling runtime changes

- As explained in Fundamentals$_{\text{section}}$:
  - Save/Restore state using Bundle$_{\text{object}}$ or Object$_{\text{object}}$
  - Handle config change manually
- Recommendation: always provide save/restore capability
  - Some config changes cannot be prevented from restarting app

Retaining an object during a configuration change

- Example - overriding onRetainNonConfigurationInstance()

```
@Override
public Object onRetainNonConfigurationInstance() {
    final MyDataObject data = collectMyLoadedData();
    return data;
}
```

- Example - Retrieving data in onCreate()

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    final MyDataObject data = (MyDataObject) getLastNonConfigurationInstance()
    if (data == null) {
        data = loadMyData();
    }
    ...
}
```

Handling the configuration change yourself

- Example - entry in Manifest:

```
<activity android:name=".MyActivity"
          android:configChanges="orientation|keyboardHidden"
          android:label="@string/app_name">
```

- Example - handling configuration change at runtime:

```
@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);

    // Checks the orientation of the screen
    if (newConfig.orientation == Configuration.ORIENTATION_LANDSCAPE) {
        Toast.makeText(this, "landscape", Toast.LENGTH_SHORT).show();
    } else if (newConfig.orientation == Configuration.ORIENTATION_PORTRAIT){
        Toast.makeText(this, "portrait", Toast.LENGTH_SHORT).show();
    }
    // Checks whether a hardware keyboard is available
    if (newConfig.hardKeyboardHidden == Configuration.HARDKEYBOARDHIDDEN_NO) {
        Toast.makeText(this, "keyboard visible", Toast.LENGTH_SHORT).show();
    } else if (newConfig.hardKeyboardHidden == Configuration.HARDKEYBOARDHIDDE
        Toast.makeText(this, "keyboard hidden", Toast.LENGTH_SHORT).show();
    }
}
```

Localization

- Use qualified resources
- Do not forget qualifier priority
- Make sure you **always** have default for **all** resources:
- If a resources is missing, app won't run
- Use layouts that will match all targeted locales
- Getting locale:

```
String locale = context.getResources().getConfiguration().locale.getDisplayNam
```

- Testing:
  - On device: Home >Menu >Settings >Locale & text >Select locale
  - In emulator:
    - adb_{tool} command line:

      ```
      setprop persist.sys.language [language code];setprop persist.sys.country [country code];st
      ```

    - Example:

      ```
      setprop persist.sys.language fr;setprop persist.sys.country CA;stop;sleep 5;start
      ```

- Testing default resources:
  - Set device/emulator to locale unsupported by your app
  - Run app
  - If app is forced to close, defaults are missing
- Doc provides localization checklist

Resource types

- There are many resource types that can be located inside 'res/'
- Documentation provides complete details of each type's:
  - Definition
  - Attributes
  - Use
  - Specificities
- Refer to doc for full details

Animation resources

- Define pre-determined animations
- 'res/anim/' = tween animations accessed through $R.anim_{class}$
  - Animation created by performing transformations on single image
- 'res/drawable/' = fram animations accessed through $R.drawable_{class}$
  - Animation created by showing in-order sequence of images

Color state list resources

- Define color resources that change based on $\text{View}_{\text{object}}$ state
- 'res/color/' accessed through $\text{R.color}_{\text{class}}$
- Example states:
  - Pressed
  - Focused
  - Enabled

Drawable resources

- Define various graphics w/ bitmaps or XML
- 'res/drawable' accessed through $R.drawable_{class}$
- Main types:

Type	Description
Bitmap file	Bitmap graph file (.png, .jpg, .gif) Creates a $BitmapDrawable_{object}$
9-patch file	PNG file w/ stretchable regions for content-based image resizing Creates a $NinePatchDrawable_{object}$
Layer list	Drawable managing other drawables Creates a $LayerDrawable_{object}$
State list	XML file referencing different bitmap graphics for different states Creates a $StateListDrawable_{object}$
Level list	Drawable managing a nbr of alternate drawables Creates a $LevelListDrawable_{object}$
Transition drawable	Drawable that cross-fades between two drawables Creates a $TransitionDrawable_{object}$
Clip drawable	Drawable that clips another drawable based on current level value Creates a $ClipDrawable_{object}$
Scale drawable	Drawable that scales another drawable based on current level value Creates a $ScaleDrawable_{object}$
Shape drawable	Defines geometric shape, incl. colors and gradients Creates a $ShapeDrawable_{object}$

Layout resources

- Define application UI layouts
- 'res/layout/' accessed through $R.layout_{class}$

Menu resources

- Define menu layouts
- 'res/menu/' accessed through $R.menu_{class}$

String resources

- Define strings, string arrays, and plurals (including formatting and styling)
- 'res/values/' accessed through $\text{R.string}_{\text{class}}$, $\text{R.array}_{\text{class}}$ and $\text{R.plurals}_{\text{class}}$
- Usually: 'res/values/strings.xml', but filename is arbitrary.

Formatting and styling

- Escape apostrophes and quotes by:
  - Nest apostrophe-containing strings within quotes (””)
  - Used ”\” to ”espace” apostrophe
  - Example:

```
<string name="good_example">"This'll work"</string>
<string name="good_example_2">This\'ll also work</string>
<string name="bad_example">This doesn't work</string>
<string name="bad_example_2">XML encodings don&apos;t work</string>
```

- Formatting strings / arguments in strings
  - Example - declaration:

```
<string name="welcome_messages">Hello, %1$s! You have %2$d new messages.</string>
```

  - Example - use in code:

```
Resources res = getResources();
String text = String.format(res.getString(R.string.welcome_messages), username, mailCount);
```

- Styling with HTML markup
  - Supported elements: <b>, <i>, <u>
  - Example - declaration:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="welcome">Welcome to <b>Android</b>!</string>
</resources>
```

  - For formatting text w/ HTML, escape the "<" with "<"
  - Example - declaration:

```
<resources>
  <string name="welcome_messages">Hello, %1$s! You have &lt;b>%2$d new messages&lt;/b>.</string>
</resources>
```

  - Example - use in code:

```
Resources res = getResources();
String text = String.format(res.getString(R.string.welcome_messages), username, mailCount);
CharSequence styledText = Html.fromHtml(text);
```

Style resource

- Define UI look and format
- 'res/values/' accessed through R.style$_{class}$
- OOS: See "Applying Styles and Themes" and "Style Resource"

More resource types

- Usually stored in 'res/values/*.xml'
- Filename is arbitrary
- All are XML-defined resources
- Nested in `<resources>` element

Type	Description
`<bool>`	Boolean value
`<color>`	Color value
`<dimen>`	Dimension value (more below)
`<item type="id">`	Unique ID for app resources
`<integer>`	Integer value
`<integer-array>`	Array of integers
`<array>`	$\text{TypedArray}_{\text{object}}$, array of "stuff"

- "ID" can be used instead of "@+id", which creates IDs on the fly

Dimensions

Unit	Description
dp/dip	Density-indepent pixel, relative 160dpi screen dp-to-pixel ratio will change w/ screen density Recommended measurement
sp	Scale-independent pixel Like "dp", but scale by font prefs Prefered measurement for fonts
pt	Points, 1/72 of an inch - based on screen size
px	Pixels Not recommended as density is non-uniform
mm	Millimeteres - based on screen size
in	Inches - based on screen size

Intents and Intent Filters

Opersys inc.

August 12, 2012

Introduction

- Activities, Services and Broadcast Receivers are Intent-activated
- Intents allow late run-time binding
- $\text{Intent}_{\text{object}}$ = passive data structure
- Intent content = description of operation
- Intent delivery (i.e. passed to):
  - Activity:
    - Context.startActivity()
    - Activity.startActivtyForResult()
  - Service:
    - Context.startService()
    - Context.bindService()
  - Broadcast Receiver:
    - Context.sendBroadcast()
    - Context.sendOrderedBroadcast()
    - Context.sendStickyBroadcast()
- Intent delivery constrained to component type
  - IOW, an Intent sent to an Activity will only be sent to an Activity

Intent objects

- Contains information relevant to:
  - Target Component
  - System
- Specifically, can contain:

Field	Description
Component Name	$\text{ComponentName}_{\text{object}}$ = fully qualified target class name + package name set in Manifest
Action	Actual action to be performed (like a fct name determines method invoked). Ex: ACTION_CALL, ACTION_MAIN, ACTION_SYNC, ACTION_BATTERY_LOW, ... Determines rest of Intent structure, esp. $\text{data}_{\text{field}}$ and $\text{extras}_{\text{field}}$
Data	URI and MIME type of data to be acted on URI: setData() and getData() MIME: setType() and getType() Setting both: setDataAndType()
Category	Kind of Component that should handle Intent Ex.: CATEGORY_BROWSABLE, CATEGORY_GADGET, CATEGORY_LAUNCHER, ...
Extras	Key-pair values for more information for component handling intent
Flags	Various flags. Defined in Intent class

Intent resolution

- Two types of Intents:
  - Explicit: Designate target through "Component Name" field
    Typically for in-app messages
  - Implicit: Do not designate target
    Typically for activating Components of other apps
- Intent filters:
  - Components w/ filter can received implicit and explicit Intents
  - Components w/o filter can only receive implicit Intents
- Intent aspects consulted for filter matching:
  - Action
  - Data (URI & MIME type)
  - Category

Intent filters

- Filters describe Intents that Component is willing to receive
- Explicit Intents delivered as-is, regardless of filter
- Filters are statically-declared in Manifest
- Filter can specify:
  - Action
  - Data
  - Category
- Filter matches only if all three match
- Do not have to specificy all three
- If Intent matches more than one Activity or Service:
  user may be prompted to select Component

- Action test:
  - Filter must contain at least one action. Otherwise all Intents blocked.
  - Intents w/o action will pass test

```
<intent-filter . . . >
    <action android:name="com.example.project.SHOW_CURRENT" />
    <action android:name="com.example.project.SHOW_RECENT" />
    <action android:name="com.example.project.SHOW_PENDING" />
    . . .
</intent-filter>
```

- Category test:
  - Must match **all** categories in Intent
  - Can list more, but cannot omit any

```
<intent-filter . . . >
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
    . . .
</intent-filter>
```

- Data test:
  - Only URI parts mentioned in filter are compared
  - Wildcards paths can be used
  - MIME-type filtering more common than URI-based filtering
  - Ex: “text/*” or “audio/*”
  - See doc for specific URI/MIME combo matching policy

```
<intent-filter . . . >
    <data android:mimeType="video/mpeg" android:scheme="http" . . . />
    <data android:mimeType="audio/mpeg" android:scheme="http" . . . />
    . . .
</intent-filter>
```

Common cases

- Most common case are MIME type filters:
  ```
  <data android:mimeType="image/*" />
  ```
- In some cases, the URI is relevant:
  ```
  <data android:scheme="http" android:type="video/*" />
  ```

Using Intent matching

- System populates Launcher using Activities that match:
  - Action: android.intent.action.MAIN
  - Category: android.intent.category.LAUNCHER
- System populates home screen using Activities that match:
  - Category: android.intent.category.HOME
- $\text{PackageManager}_{\text{object}}$ has methods for:
  - Returning all Components that can accept a particular Intent:
    - queryIntentActivities()
    - queryIntentServices()
    - queryIntentReceivers()
  - Determining the best Component to respond to an Intent:
    - resolveIntentActivities()
    - resolveIntentServices()
    - resolveIntentReceivers()

Data Storage

Opersys inc.

August 12, 2012

Storage options

Shared preferences	Private primitive key-pair values
Internal storage	Private data on device memory
External storage	Public data on shared external device (SD)
SQLite DB	Private DB
Network connection	Web-based storage (REST)

Using Shared Preferences

- Use SharedPreferences$_{class}$
- Primitive data: booleans, floats, ints, longs, and strings
- Data will persist accross user sessions and lifecycle
- To get a SharePreferences$_{object}$:
  - getSharedPreferences(): Allows having separate preference files for Activity (1st param)
  - getPreferences(): Allows having single preference file
- Also look at PreferenceActivity$_{class}$
- To write values:
  1. Call edit() to get a SharedPreferences.Editor$_{object}$
  2. Add values w/ methods like putBoolean() and putString()
  3. Commit the new values w/ commit()
- To read values: use methods like getBoolean() and getString()

Example preference saving and restoring

```
public class Calc extends Activity {
    public static final String PREFS_NAME = "MyPrefsFile";

    @Override
    protected void onCreate(Bundle state){
       super.onCreate(state);
       . . .

       // Restore preferences
       SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
       boolean silent = settings.getBoolean("silentMode", false);
       setSilent(silent);
    }

    @Override
    protected void onStop(){
       super.onStop();

      // We need an Editor object to make preference changes.
      // All objects are from android.context.Context
      SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
      SharedPreferences.Editor editor = settings.edit();
      editor.putBoolean("silentMode", mSilentMode);

      // Commit the edits!
      editor.commit();
    }
```

Using Internal Storage

- Can save files on internal storage
- File storage is app-specific (no shared files)
- Upon uninstall, files are deleted
- Writing private file:
  1. Provide filename to openFileOutput(), returns $\text{FileOutputStream}_{\text{object}}$
  2. Write to file w/ write()
  3. Close stream w/ close()
- Example:

```
String FILENAME = "hello_file";
String string = "hello world!";

FileOutputStream fos = openFileOutput(FILENAME, Context.MODE_PRIVATE);
fos.write(string.getBytes());
fos.close();
```

- File modes:
  - MODE_PRIVATE
  - MODE_APPEND
  - MODE_WORLD_READABLE
  - MODE_WORLD_WRITABLE
- File reading:
  1. Provide filename to openFileInput(), returns $\text{FileInputStream}_{\text{object}}$
  2. Read bytes using read()
  3. Close stream w/ close()
- Can use 'res/raw' to store files at build time. Use openRawResource() w/ `'R.raw.<filename>'`, will return $\text{InputStream}_{\text{object}}$.
- Cache files: use getCacheDir() to open $\text{File}_{\text{object}}$. Will be erased on low-storage.
- Other functions:
  - getFilesDir(): own internal files absolute fs path
  - getDir(): create/open dir within app internal storage
  - deleteFile(): deletes file
  - fileList(): returns array of file currently stored by app

Using External Storage

- All Android-compatible devices have external storage, typically SD
- Files storage on external storage are: world-readable and world-writeable
- Files visible when device plugged in through USB
- Use getExternalStorageState() to check the media's state:

```
boolean mExternalStorageAvailable = false;
boolean mExternalStorageWriteable = false;
String state = Environment.getExternalStorageState();

if (Environment.MEDIA_MOUNTED.equals(state)) {
    // We can read and write the media
    mExternalStorageAvailable = mExternalStorageWriteable = true;
} else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
    // We can only read the media
    mExternalStorageAvailable = true;
    mExternalStorageWriteable = false;
} else {
    // Something else is wrong. It may be one of many other states, but all we need
    //  to know is we can neither read nor write
    mExternalStorageAvailable = mExternalStorageWriteable = false;
```

- Accessing files on external storage
  - API level 8 and +, use getExternalFilesDir()
  - API level 7 and -, use getExternalStorageDirectory() and use app-specific dir
- Saving files on external storage that should be shared
  - API level 8 and +, use getExternalStoragePublicDirectory()
  - API level 7 and -, use getExternalStorageDirectory()
- Saving cache files
  - API level 8 and +, use getExternalCacheDir()
  - API level 7 and -, use getExternalStorageDirectory()

Using Databases

- Based on SQLite
- Data private to all Components in app, but not other apps
- Can be used in Activity
- Use $\text{SQLiteOpenHelper}_{\text{object}}$ and override onCreate() to create tables:

```
public class DictionaryOpenHelper extends SQLiteOpenHelper {

    private static final int DATABASE_VERSION = 2;
    private static final String DICTIONARY_TABLE_NAME = "dictionary";
    private static final String DICTIONARY_TABLE_CREATE =
                "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" +
                KEY_WORD + " TEXT, " +
                KEY_DEFINITION + " TEXT);";

    DictionaryOpenHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(DICTIONARY_TABLE_CREATE);
    }
}
```

- Get $\text{SQLiteOpenHelper}_{\text{object}}$ from constructor
- Write: getWriteableDatabase(), returns $\text{SQLiteDatabase}_{\text{object}}$
- Read: getReadableDatabase(), returns $\text{SQLiteDatabase}_{\text{object}}$
- Execute query: SQLiteDatabase.query(), returns $\text{Cursor}_{\text{object}}$
- Use $\text{Cursor}_{\text{object}}$ to navigate results
- Build complex query: $\text{SQLiteQueryBuilder}_{\text{object}}$
- Use $\text{sqlite3}_{\text{tool}}$ to debug DB

Using a Network Connection

- java.net.*
- android.net.*

Data Backup

- $\text{Backup}_{\text{service}}$ can backup data to "cloud"
- Not guaranteed to be available on all devices
- Restore on factory reset or activation of new Android device
- Transparent process
- On backup, Backup Manager queries apps for backup data
- On restore, Backup Manager returns data to app
- Not an API. Can only respond to Backup Manager, not query.
- To use this service, must implement a backup agent
- OOS: See "Data Backup"

Content Providers

Opersys inc.

August 12, 2012

Introduction

- Only way to share data accross apps
- Android ships w/ default Content Providers for audio, video, images, contacts, etc. (see android.provider package)
- Some Content Providers require permissions to speak to
- Can create own Content Provider
- Can add data to existing Content Provider
- Actual storage is opaque to Content Provider user

Content Provider basics

- Use ContentResolver$_{object}$ to talk to Content Provider:
  `ContentResolver cr = getContentResolver();`
- Client never deals w/ ContentProvider$_{object}$
- On query, system:
  - Identifies Content Provider
  - Makes sure it's running
- Typical config: N clients, 1 Content Provider
- Data model:
  - Simple tables, like, w/ rows and columns
  - Unique ID for each record/row
- URIs:
  - Each Content Provider has unique URI for its dataset
  - All Content Provider URIs start w/: "content://"

Querying a Content Provider

- Need:
  - Content Provider's URI
  - Queried data field names
  - Queried data field types
  - If querying record, need its ID
- Query methods:
  - ContentResolver.query()
  - Activity.managedQuery() - automagically handles lifecycle
- Both return $\text{Cursor}_{\text{object}}$ for reading result
- Query arguments:
  - URI
  - Data column names - "null" means all
  - Row filter - format is SQL "WHERE" w/o "WHERE" word, "null" means all
  - Selection arguments
  - Sorting order - format is SQL "ORBER BY" w/o "ORDER BY" words, "null" means default

Example query on one ID - ID appended to URI:

```
import android.provider.Contacts.People;
import android.content.ContentUris;
import android.net.Uri;
import android.database.Cursor;

// Use the ContentUris method to produce the base URI for the contact with _ID == 2
Uri myPerson = ContentUris.withAppendedId(People.CONTENT_URI, 23);

// Alternatively, use the Uri method to produce the base URI.
// It takes a string rather than an integer.
Uri myPerson = Uri.withAppendedPath(People.CONTENT_URI, "23");

// Then query for this specific record:
Cursor cur = managedQuery(myPerson, null, null, null, null);
```

Example multi-argument query:

```
import android.provider.Contacts.People;
import android.database.Cursor;

// Form an array specifying which columns to return.
String[] projection = new String[] {
                              People._ID,
                              People._COUNT,
                              People.NAME,
                              People.NUMBER
                            };

// Get the base URI for the People table in the Contacts content provider.
Uri contacts =  People.CONTENT_URI;

// Make the query.
Cursor managedCursor = managedQuery(contacts,
                          projection, // Which columns to return
                          null,       // Which rows to return (all rows)
                          null,       // Selection arguments (none)
                          // Put the results in ascending order by name
                          People.NAME + " ASC");
```

Reading retrieved data

- $\text{Cursor}_{\text{object}}$ contains zero or more records
- Must read appropriate type from each column
- If data is too large, cell will hold content URI. Must use ContentResolver.openInputStream() to extract.

```
import android.provider.Contacts.People;

private void getColumnData(Cursor cur){
    if (cur.moveToFirst()) {

        String name;
        String phoneNumber;
        int nameColumn = cur.getColumnIndex(People.NAME);
        int phoneColumn = cur.getColumnIndex(People.NUMBER);
        String imagePath;

        do {
            // Get the field values
            name = cur.getString(nameColumn);
            phoneNumber = cur.getString(phoneColumn);

            // Do something with the values.
            ...

        } while (cur.moveToNext());

    }
```

Modifying data

- Data can be modifiedy by:
  - Adding new records
  - Adding new values to existing records
  - Batch updating existing records
  - Deleting records
- Use ContentResolver methods
- May require permissions for writing

Adding records using ContentValues$_{object}$

```
import android.provider.Contacts.People;
import android.content.ContentResolver;
import android.content.ContentValues;

ContentValues values = new ContentValues();

// Add Abraham Lincoln to contacts and make him a favorite.
values.put(People.NAME, "Abraham Lincoln");
// 1 = the new contact is added to favorites
// 0 = the new contact is not added to favorites
values.put(People.STARRED, 1);

Uri uri = getContentResolver().insert(People.CONTENT_URI, values);
```

Adding new values

```
Uri phoneUri = null;
Uri emailUri = null;

// Add a phone number for Abraham Lincoln.  Begin with the URI for
// the new record just returned by insert(); it ends with the _ID
// of the new record, so we don't have to add the ID ourselves.
// Then append the designation for the phone table to this URI,
// and use the resulting URI to insert the phone number.
phoneUri = Uri.withAppendedPath(uri, People.Phones.CONTENT_DIRECTORY);

values.clear();
values.put(People.Phones.TYPE, People.Phones.TYPE_MOBILE);
values.put(People.Phones.NUMBER, "1233214567");
getContentResolver().insert(phoneUri, values);

// Now add an email address in the same way.
emailUri = Uri.withAppendedPath(uri, People.ContactMethods.CONTENT_DIRECTORY);

values.clear();
// ContactMethods.KIND is used to distinguish different kinds of
// contact methods, such as email, IM, etc.
values.put(People.ContactMethods.KIND, Contacts.KIND_EMAIL);
values.put(People.ContactMethods.DATA, "test@example.com");
values.put(People.ContactMethods.TYPE, People.ContactMethods.TYPE_HOME);
getContentResolver().insert(emailUri, values);
```

- If content too large, put content URI for data and use ContentResolver.openOutputStream()

Batch updating and deleting

- Batch updating: call ContentResolver.update() w/ columns and values to change
- Deleting, using ContentResolver.delete():
  - Single row: provide URI for specific row
  - Multiple rows: provide URI of record type and SQL WHERE clause

Creating a Content Provider

- Set up storage system
- Extend ContentProvider$_{\text{class}}$
- Declare Content Provider in Manifest

Extending the ContentProvider$_{class}$

- Implement:

```
query()
insert()
update()
delete()
getType()
onCreate()
```

- **All** methods must be thread-safe
- query() must return Cursor$_{object}$
- "Cursor" is actually interface. Objects that can be used: SQLiteCursor, MatrixCursor
- Strongly recommended to call ContentResolver.notifyChange() when data changes

Recommendations for client usability and class accessability

- Define CONTENT_URI:

```
public static final Uri CONTENT_URI =
                Uri.parse("content://com.example.codelab.transportationprovider
```

- Define CONTENT_URI for subtables:

```
content://com.example.codelab.transportationprovider/train
content://com.example.codelab.transportationprovider/air/domestic
content://com.example.codelab.transportationprovider/air/international
```

- Define and provide static strings for column names, and document type
- Provide "_id" column that has unique ID
- If using SQLite, make sure to use AUTOINCREMENT:

```
INTEGER PRIMARY KEY AUTOINCREMENT
```

- If handling new type, define the new MIME type

- If exposing large data, put content URI in cell and provide _data$_{field}$ in record to indicate filesystem location of resource for use by ContentResolver. Latter will automagically request _data$_{field}$ when client calls ContentResolver.openInputStream(). Since Content Resolver has higher priority than client, it will be authorized to read your file and provide it to the client.

Declaring the Content Provider

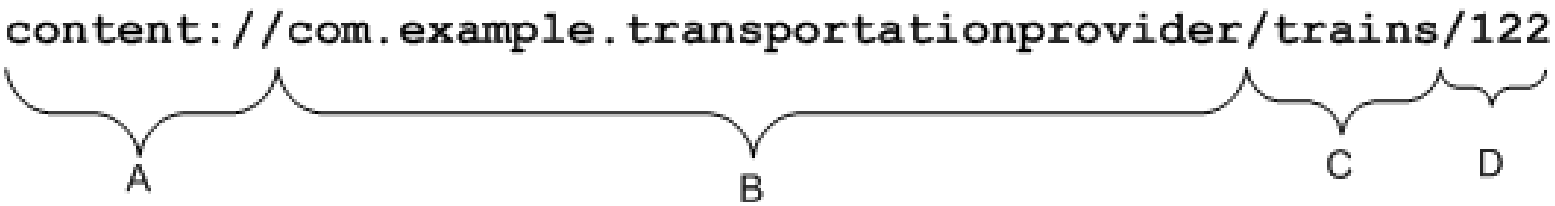
- Use <provider> in Manifest
- Use $\text{authorities}_{\text{attribute}}$ to specify content URI identifying this Content Provider

```
<provider android:name="com.example.autos.AutoInfoProvider"
          android:authorities="com.example.autos.autoinfoprovider"
          . . . />
</provider>
```

- Note that paths are not specified, just the root:

```
content://com.example.autos.autoinfoprovider/honda
content://com.example.autos.autoinfoprovider/gm/compact
content://com.example.autos.autoinfoprovider/gm/suv
```

Content URI summary



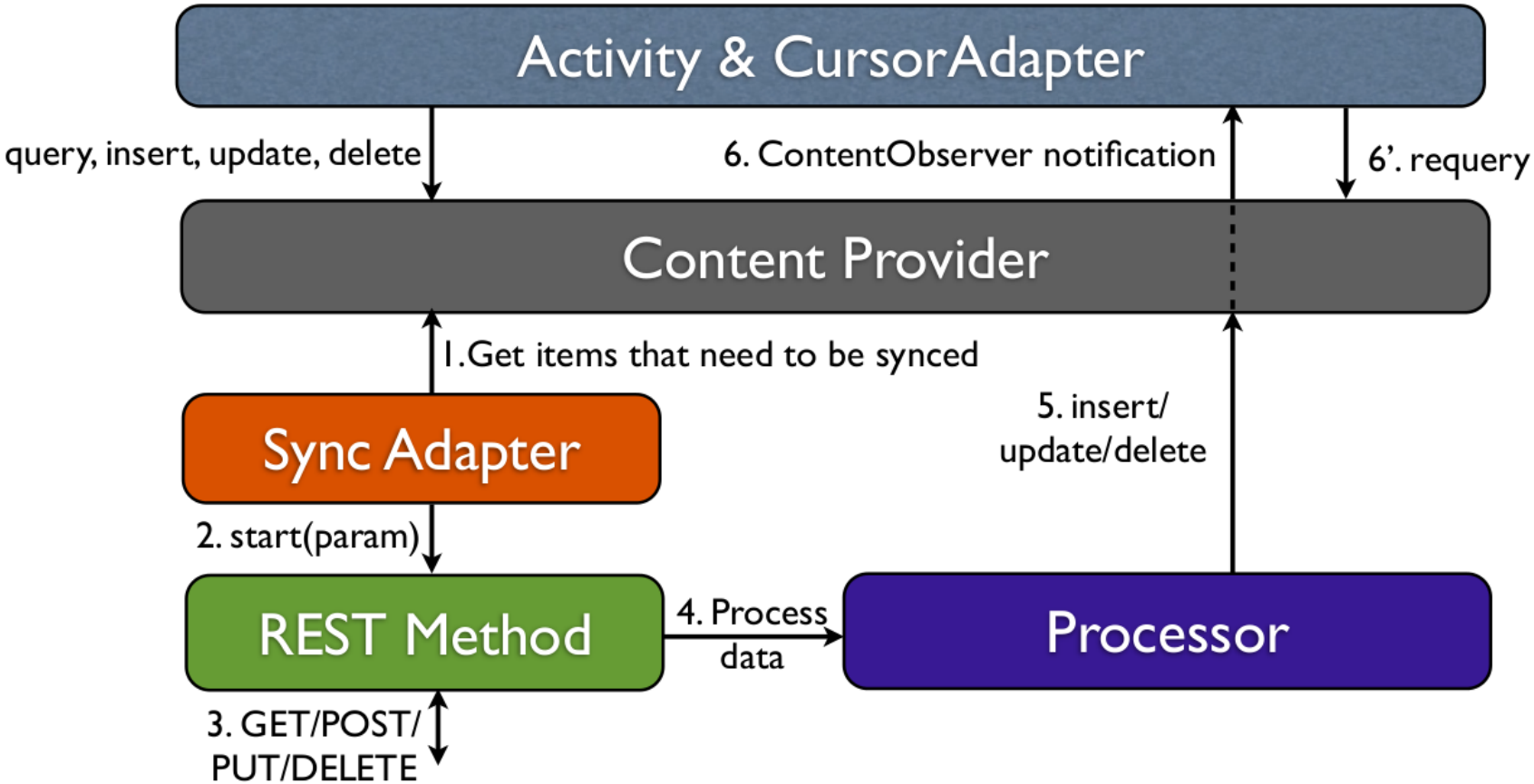
A. Std prefix. Never modified.
B. Authority. Should be fully-qualified class name.
C. Path indicating data type
D. Record ID, if relevant

REST-based Applications

Opersys inc.

August 12, 2012

Architecture



Sync Adapter

- Enables automagic sync function callback
- Introduced in 2.2 - Froyo
- Gates/Calls back to onPerformSync
- User controls behavior in Settings-Accounts&Sync
- Heavily tied to AccountManager
- AccountManager provides credentials screen
- Implement AbstractThreadedSyncAdapter
- Declare Sync Adapter as Service in Manifest

Guidelines

- Refer to and/or copy Sync Adapter example
- Don't forget to put syncadapter.xml in 'res/xml'
- Use Apache http calls, not java ones
- Don't forget to include all necessary permissions in Manifest

References

- http://www.google.com/events/io/2010/sessions/developing-RESTful-android-apps.html
- http://developer.android.com/resources/samples/SampleSyncAdapter/ir
- http://ericmiles.wordpress.com/2010/09/22/connecting-the-dots-with-android-syncadapter/

Security and Permissions

Opersys inc.

August 12, 2012

Introduction

- Most security enforced at process level: UID, GID
- Permissions enforce restrictions on:
  - Per-process operations
  - Per-URI access

Security architecture

- Applications are sandboxed
- Specific permissions required to “exit” sandbox
- Decision to grant access based on:
  - Certificates
  - User prompts
- All permissions must be declared **statically**

Application signing

- All apps must be signed with private key
- No CA need be involved
- Impact of signatures:
  - Permissions
  - UID sharing

User IDs and file access

- Each app has unique UID
- UID assigned at install time
- UID does not change over time
- Use $\text{sharedUserId}_{\text{attribute}}$ in Manifest to ensure common UID
- Only 2 apps signed w/ same key can have same UID
- Can use MODE_WORLD_READABLE and MODE_WORLD_WRITABLE when creating new file w/:
  - getSharedPreferences()
  - openFileOutput()
  - openOrCreateDatabase()

User permissions

- Must use <uses-permission> in Manifest to claim permissions

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.android.app.myapp" >
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    ...
</manifest>
```

- Permissions granted at install time based on:
  - Checks against signature, and/or
  - Interaction w/ user
- There are no runtime checks
- If fail, will get SecurityException at runtime
- Permissions checked when:
  - Calling a system function
  - Starting an Activity
  - Sending or receiving Broadcasts
  - Accessing or opening a Content Provider
  - Binding or starting a Service
- May define and enforce permissions for your own apps

Declaring and enforcing permissions

- Must use <permission> in Manifest to define your own permissions

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.me.app.myapp" >
    <permission android:name="com.me.app.myapp.permission.DEADLY_ACTIVITY"
        android:label="@string/permlab_deadlyActivity"
        android:description="@string/permdesc_deadlyActivity"
        android:permissionGroup="android.permission-group.COST_MONEY"
        android:protectionLevel="dangerous" />
    ...
</manifest>
```

- Required: $\text{protectionLevel}_{\text{attribute}}$ = how to inform user — who is allowed
- Optional: $\text{permissionGroup}_{\text{attribute}}$ = for simplfying permission UI

- Use $\text{label}_{\text{atttribute}}$ and $\text{description}_{\text{attribute}}$ for user info purposes

```
<string name="permlab_callPhone">directly call phone numbers</string>
<string name="permdesc_callPhone">Allows the application to call
    phone numbers without your intervention. Malicious applications may
    cause unexpected calls on your phone bill. Note that this does not
    allow the application to call emergency numbers.</string>
```

- List all permissions

```
$ adb shell pm list permissions -s
All Permissions:

Network communication: view Wi-Fi state, create Bluetooth connections, full
Internet access, view network state

Your location: access extra location provider commands, fine (GPS) location,
mock location sources for testing, coarse (network-based) location

Services that cost you money: send SMS messages, directly call phone numbers

...
```

Enforcing permissions in AndroidManifest.xml

- Use $\text{permission}_{\text{attribute}}$ in Component description in Manifest
- Activity permissions restrict start. Checked at:
  - Context.startActivity()
  - Activity.startActivityForResult()
- Service permissions restrict start and bind. Checked at:
  - Context.startService()
  - Context.stopService()
  - Context.bindService()
- Broadcast Receiver permissions restrict send/received. Checked/supplied at:
  - Context.sendBroadcast() - checked **after** call
  - Context.registerReceiver() - control who can broadcast
  - Context.sendBroadcast() - control who can receive

- Content Provider permissions restrict data access
- $\text{readPermission}_{\text{attribute}}$ $\text{writePermission}_{\text{attribute}}$
  - ContentResolver.query() requires read
  - ContentResolver.insert() requires write
  - ContentResolver.update() requires write
  - ContentResolver.delete() requires write
- “write” does not imply “read”

Enforcing permissions when sending Broadcasts

- Call Context.sendBroadcast() with permission string

Other permission enforcement

- Use Context.checkCallingPermission() to check caller's rights
- Use Context.checkPermission() to check permission against PID
- Use PackageManager.checkPermission() to check package's permissions

URI permissions

- Sometimes need to dissociate data sets
- Ex.: Email vs. opening their attachments
- Use Intent flags:
  - FLAG_GRANT_READ_URI_PERMISSION
  - FLAG_GRANT_WRITE_URI_PERMISSION
- Use grantUriPermissions$_{\text{attribute}}$ or `<grant-uri-permissions>`

Remote Interfaces

Opersys inc.

August 12, 2012

Introduction

- Object passing between processes
- Must decompose objects so they can be marshalled accross
- $\text{aidl}_{\text{tool}}$ generates marshalling code
- AIDL = Android Interface Definition Language
- IPC mechanism is interface-based like COM or Corba
- Uses proxy class to pass values

Implementing IPC using AIDL

1. Create an .aidl$_{file}$
2. Add .aidl$_{file}$ to Makefile (automagically managed by Eclipse)
3. Implement the interface method:
   Extend the aidl-generated class and implement methods
4. Expose the interface to clients:
   Return subclass instance from Service.onBind(Intent)

Create an .aidl file

- Declare one or more methods
- Methods can take many parameters and return values
- Must import **all** non-built-in types
- AIDL-supported data types:
  - Primitive Java language types - no import needed
  - The following classes - no import needed:
    - String
    - List
    - Map
    - CharSequence
  - Other AIDL-generated interfaces - import needed
  - Custom classes implement the Parcealable protocol - import needed

AIDL syntax

```
// My AIDL file, named SomeClass.aidl
// Note that standard comment syntax is respected.
// Comments before the import or package statements are not bubbled up
// to the generated interface, but comments above interface/method/field
// declarations are added to the generated interface.

// Include your fully-qualified package statement.
package com.android.sample;

// See the list above for which classes need
// import statements (hint--most of them)
import com.android.sample.IAtmService;

// Declare the interface.
interface IBankAccountService {

    // Methods can take 0 or more parameters, and
    // return a value or void.
    int getAccountBalance();
    void setOwnerNames(in List<String> names);

    // Methods can even take other AIDL-defined parameters.
    BankAccount createAccount(in String name, int startingDeposit, in IAtmService atmService);

    // All non-Java primitive parameters (e.g., int, bool, etc) require
    // a directional tag indicating which way the data will go. Available
    // values are in, out, inout. (Primitives are in by default, and cannot be otherwise).
    // Limit the direction to what is truly needed, because marshalling parameters
    // is expensive.
    int getCustomerList(in String branch, out String[] customerList);
}
```

Implementing the interface

- AIDL generates interface file with same name as .aidl$_{file}$
- Eclipse manages this automagically - just put .aidl$_{file}$ w/ Java$_{files}$
- Generated interface contains "Stub" inner class
- Extend the foo.Stub$_{class}$ and implement mehtods
- Example:

```
// No need to import IRemoteService if it's in the same project.
private final IRemoteService.Stub mBinder = new IRemoteService.Stub(){
    public int getPid(){
        return Process.myPid();
    }
}
```

- Some rules:
  - No exceptions thrown will be sent to caller
  - All IPC call are synchronous, use threads if needed
  - Cannot declare static fields, only methods

Exposing your interface to clients

- Implement Service.onBind(Intent)
- Return instance of class that implements interface
- Example:

```
public class RemoteService extends Service {
...
    @Override
    public IBinder onBind(Intent intent) {
        // Select the interface to return.  If your service only implements
        // a single interface, you can just return it here without checking
        // the Intent.
        if (IRemoteService.class.getName().equals(intent.getAction())) {
            return mBinder;
        }
...
        return null;
    }

    /**
     * The IRemoteInterface is defined through IDL
     */
    private final IRemoteService.Stub mBinder = new IRemoteService.Stub() {
        public void registerCallback(IRemoteServiceCallback cb) {
            if (cb != null) mCallbacks.register(cb);
        }
        public void unregisterCallback(IRemoteServiceCallback cb) {
            if (cb != null) mCallbacks.unregister(cb);
        }
    };
```

Pass by value parameters using parcelables

- Passing classes to other processes
- Must ensure class code available to caller
- Usually caller/callee are Activity & Service from same app
- Parts of the $\text{Parcelable}_{\text{protocol}}$:
  1. Make class implement $\text{Parcelable}_{\text{interface}}$
  2. Implement `public void writeToParcel(Parcel out)`
     Takes current object state and writes it to parcel
  3. Add static $\text{CREATOR}_{\text{field}}$ to class which is object implementing $\text{Parcelable.Creator}_{\text{interface}}$
  4. Create .aidl file declaring parcelable class

Example implementation of Parcelable$_{protocol}$ I

```
import android.os.Parcel;
import android.os.Parcelable;

public final class Rect implements Parcelable {
    public int left;
    public int top;
    public int right;
    public int bottom;

    public static final Parcelable.Creator<Rect> CREATOR = new Parcelable.Creator<Rect>() {
        public Rect createFromParcel(Parcel in) {
            return new Rect(in);
        }

        public Rect[] newArray(int size) {
            return new Rect[size];
        }
    };

    public Rect() {
    }

    private Rect(Parcel in) {
        readFromParcel(in);
    }
```

Example implementation of $\text{Parcelable}_{\text{protocol}}$ II

```
    public void writeToParcel(Parcel out) {
        out.writeInt(left);
        out.writeInt(top);
        out.writeInt(right);
        out.writeInt(bottom);
    }

    public void readFromParcel(Parcel in) {
        left = in.readInt();
        top = in.readInt();
        right = in.readInt();
        bottom = in.readInt();
    }
}
```

- Corresponding Rect.aidl:

```
package android.graphics;

// Declare Rect so AIDL can find it and knows that it implements
// the parcelable protocol.
parcelable Rect;
```

- See $\text{Parcel}_{\text{class}}$ for more details

Calling an IPC method

- Steps to call remote interface:
  1. Declare interface of same type as that declared in .aidl file
  2. Implement $\text{ServiceConnection}_{\text{interface}}$
  3. Call Context.bindService(), passing it $\text{ServiceConnection}_{\text{interface}}$ implementation
  4. Use foo.Stub.asInterface((IBinder)service) in ServiceConnection.onServiceConnected() to cast "service" to foo_{type}
  5. Call methods on your interface
     Trap $\text{DeadObjectException}_{\text{exceptions}}$ - thrown if connection lost
  6. Call Context.unbindService() to disconnect
- Objects are reference counted accross processes
- Can send anonymous objects as method arguments
- See documentation for full example

AndroidManifest.xml

Opersys inc.

August 12, 2012

Introduction

- All apps must have “AndroidManifest.xml” file
- Names the Java package
- Describes the app Components
- Determines which processes will host app Components
- Declares permissions required by app
- Declares permissions required by other apps to access its Components
- Lists instrumentation classes providing profiling and other information
- Declares the minimum API level required by app
- Lists libraries app must be linked against
  - See <uses-library>entry in Manifest explanation
  - See “Working with Library Projects” of “Developing in Eclipse with ADT”

Structure of the Manifest file I

```
<?xml version="1.0" encoding="utf-8"?>

<manifest>

    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />

    <application>

        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>

        <activity-alias>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </activity-alias>
```

Structure of the Manifest file II

```
        <service>
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>

        <receiver>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </receiver>

        <provider>
            <grant-uri-permission />
            <meta-data />
        </provider>

        <uses-library />

    </application>

</manifest>
```

Complete list of legal elements

Element	Description
<action>	Adds action to $\text{Intent}_{\text{filter}}$
<activity>	Declares an Activity
<activity-alias>	Declares an alias for an Activity
<application>	Declaration of the application
<category>	Adds category to $\text{Intent}_{\text{filter}}$
<data>	Adds data specification to $\text{Intent}_{\text{filter}}$
<grant-uri-permission>	Specifies Content Provider data subset permissions
<instrumentation>	Declares $\text{Instrumentation}_{\text{class}}$ for monitoring app
<intent-filter>	Specifies types of Intents that Component can respond to
<manifest>	Root element of Manifest file
<meta-data>	Name-value pair data supplied to parent Component
<path-permission>	Defines path and permissions for specific data subset within Content Provider
<permission>	Declares permissions required to access Components of this app
<permission-group>	Declares name for logical grouping of related permissions
<permission-tree>	Declares base name for a permissions tree
<provider>	Declares a Content Provider
<receiver>	Declares a Broadcast Receiver
<service>	Declares a Service
<supports-screens>	Specifies which screen dimensions are supported by app
<uses-configuration>	Indicates which hardware and software features are required by app
<uses-feature>	Declares specific features used by app. Non-binding; used by Market.
<uses-library>	Specifies shared library that app must be linked against
<uses-permission>	Request permission required to run app
<uses-sdk>	Enables specifying platform level compatibility

File conventions

- Elements
  - Only <manifest>and <application>are required
  - All values set through attributes, not character data within element
  - Elements at same level are generally unordered
- Attributes
  - Typically optional
  - Common attributes begin with “android:” prefix
- Declaring class names
  - Use $\text{name}_{\text{attribute}}$
  - Can use “.” as first character in string as shorthand for package name
  - Equivalents:

```
<service android:name="com.example.project.SecretService" . . . >
<service android:name=".SecretService" . . . >
```

- Multiple values
  - When multiple values, repeat element - not multi-value in single element
- Resource values
  - Format: @[package:]type:name
  - Example: “@drawable/smallPic”
- String values
  - Use double backslashes “\\” to escape characters

Device Administration

Opersys inc.

August 12, 2012

Introduction

- Starting Android 2.2
- Security enforcement
- NOT application deployment or control
- Includes remote-wipe capability

Device administration API overview - types of apps

- Email clients (ex: Exchange)
- Security applications that do remote wipe
- Device management services and applications

How does it work?

- Sysadmin writes policy-enforcing app: hard-coded or dynamic
- App is installed on device - no automated provisioning yet:
  - Android Market
  - Enabling non-market installation
  - Distributed app through other means, email or web
- System prompts user to enable device admin app (implementation-specific)
- Once enabled, policies in force. Typically, accepting confers benefits.
- What if?:
  - App not enabled:
    App remains inactive on device, user doesn't get benefits.
  - User fails to comply:
    Implementation-specific. Usually: denied sync.
  - Connecting to server w/ unsupported device admin API:
    Connection not allowed. Must support all stated policies
  - Multiple admin apps:
    Strictest policy is enforced

Policies

- Password enabled (PIN incl.)
- Minimum password length
- Alphanumeric password required
- Maximum failed password attempts
- Maximum inactivity time lock

Other features

- Prompt user to set a new password
- Lock device immediately
- Wipe the device's data (restore factory defaults)

Developing a device administration application

1. Creating the Manifest
2. Implementing the code
3. Subclassing DeviceAdminReceiver
4. Managing policies

Creating the Manifest

- Must include:
  - Subclass of $\text{DeviceAdminReceiver}_{\text{class}}$ that includes:
    - BIND_DEVICE_ADMIN permission
    - $\text{ACTION\_DEVICE\_ADMIN\_ENABLED Intent}_{\text{filter}}$
  - Metadata declaration of security policies used

Sample device administration Manifest

```
<activity android:name=".app.DeviceAdminSample$Controller"
          android:label="@string/activity_sample_device_admin">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.SAMPLE_CODE" />
    </intent-filter>
</activity>

<receiver android:name=".app.DeviceAdminSample"
          android:label="@string/sample_device_admin"
          android:description="@string/sample_device_admin_description"
          android:permission="android.permission.BIND_DEVICE_ADMIN">
    <meta-data android:name="android.app.device_admin"
               android:resource="@xml/device_admin_sample" />
    <intent-filter>
        <action android:name="android.app.action.DEVICE_ADMIN_ENABLED" />
    </intent-filter>
</receiver>
```

Notes on Manifest

- Must use android:permission="android.permission.BIND_DEVICE_ADMIN" to ensure only system can call
- android.app.action.DEVICE_ADMIN_ENABLED is primary DeviceAdminReceiver$_{\text{subclass}}$ action to be handled
- On suitable Broadcast events, impose policy
- android:resource="@xml/device_admin_sample" declares security policy in metadata
  - Example:

```
<device-admin xmlns:android="http://schemas.android.com/apk/res/android">
  <uses-policies>
    <limit-password />
    <watch-login />
    <reset-password />
    <force-lock />
    <wipe-data />
  </uses-policies>
</device-admin>
```

Implementing the code - foundation classes

- DeviceAdminReceiver - convenient way for interpreting raw system Intents
- DevicePolicyManager - class for managing policies enforced on device
- DeviceAdminInfo - metadata specified for device

Subclassing DeviceAdminReceiver

- Class contains a series of callbacks triggered on particular events
- Example:

```
public class DeviceAdminSample extends DeviceAdminReceiver {

...
    @Override
    public void onEnabled(Context context, Intent intent) {
        showToast(context, "Sample Device Admin: enabled");
    }

    @Override
    public CharSequence onDisableRequested(Context context, Intent intent) {
        return "This is an optional message to warn the user about disabling.";
    }

    @Override
    public void onDisabled(Context context, Intent intent) {
        showToast(context, "Sample Device Admin: disabled");
    }

    @Override
    public void onPasswordChanged(Context context, Intent intent) {
        showToast(context, "Sample Device Admin: pw changed");
    }

    void showToast(Context context, CharSequence msg) {
        Toast.makeText(context, msg, Toast.LENGTH_SHORT).show();
    }
...
```

Enabling the application

- Most important event: user enabling app
- Explicit enabling
- Enabled on ACTION_ADD_DEVICE_ADMIN$_{\text{intent}}$

Code example

```
 private OnClickListener mEnableListener = new OnClickListener() {
    public void onClick(View v) {
        // Launch the activity to have the user enable our admin.
        Intent intent = new Intent(DevicePolicyManager.ACTION_ADD_DEVICE_ADMIN);
        intent.putExtra(DevicePolicyManager.EXTRA_DEVICE_ADMIN,
                mDeviceAdminSample);
        intent.putExtra(DevicePolicyManager.EXTRA_ADD_EXPLANATION,
                "Additional text explaining why this needs to be added.");
        startActivityForResult(intent, RESULT_ENABLE);
    }
};

...
// This code checks whether the device admin app was successfully enabled.
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    switch (requestCode) {
        case RESULT_ENABLE:
            if (resultCode == Activity.RESULT_OK) {
                Log.i("DeviceAdminSample", "Administration enabled!");
            } else {
                Log.i("DeviceAdminSample", "Administration enable FAILED!");
            }
            return;
    }
    super.onActivityResult(requestCode, resultCode, data);
```

- intent.putExtra(DevicePolicyManager, EXTRA_DEVICE_ADMIN, mDeviceAdminSample) states that mDeviceAdminSample (which is DeviceAdminReceiver) is target policy - invokes earlier screenshot
- Used DevicePolicyManager's isAdminActive() to confirm app is enabled

```
DevicePolicyManager mDPM;
...
boolean active = mDPM.isAdminActive(mDeviceAdminSample);
if (active) {
    // Admin app is active, so do some admin stuff
                ...
} else {
    // do something else
```

Managing policies

- Get handle on DevicePolicyManager:

```
DevicePolicyManager mDPM =
    (DevicePolicyManager)getSystemService(Context.DEVICE_POLICY_SERVICE);
```

- Tasks that can be performed w/ handle:
  - Set password policies
  - Set device lock
  - Perform data wipe

Set password policies

- Prompt user to set password:

```
Intent intent = new Intent(DevicePolicyManager.ACTION_SET_NEW_PASSWORD);
startActivity(intent);
```

- Set password quality:
  - PASSWORD_QUALITY_ALPHABETIC
  - PASSWORD_QUALITY_ALPHANUMERIC
  - PASSWORD_QUALITY_NUMERIC
  - PASSWORD_QUALITY_SOMETHING
  - PASSWORD_QUALITY_UNSPECIFIED
- Example:

```
DevicePolicyManager mDPM;
ComponentName mDeviceAdminSample;
...
mDPM.setPasswordQuality(mDeviceAdminSample, DevicePolicyManager.PASSWORD_QUALITY_ALPHANUMERIC);
```

- Set minimum password length

```
DevicePolicyManager mDPM;
ComponentName mDeviceAdminSample;
int pwLength;
...
mDPM.setPasswordMinimumLength(mDeviceAdminSample, pwLength);
```

- Set maximum failed password attempts

```
DevicePolicyManager mDPM;
ComponentName mDeviceAdminSample;
int maxFailedPw;
 ...
mDPM.setMaximumFailedPasswordsForWipe(mDeviceAdminSample, maxFailedPw);
```

Set device lock

- Maximum period of user inactivity before device lock

```
DevicePolicyManager mDPM;
ComponentName mDeviceAdminSample;
...
long timeMs = 1000L*Long.parseLong(mTimeout.getText().toString());
mDPM.setMaximumTimeToLock(mDeviceAdminSample, timeMs);
```

- Immediate lock

```
DevicePolicyManager mDPM;
mDPM.lockNow();
```

Perform data wipe

- Use wipeData()

```
DevicePolicyManager mDPM;
mDPM.wipeData(0);
```

Native Development

Opersys inc.

August 12, 2012

Introduction

- Use NDK, companion to SDK
- Useful for:
  - Porting existing body of code to Android
  - Developing optimized native apps, especially for gaming
- Provides:
  - Tools and build files to generate native code libraries from C/C++
  - Way to embed native libs into .apk
  - Set of stable (forward-compatible) native libs
  - Documentation, samples and tutorials
- Enables:
  - Calling native code from Java using JNI
  - Implementing fully native apps (since 2.3)
- Instruction set:
  - ARMv5TE = baseline
  - ARMv7 supported on some devices
  - Can build for both archs and ship in same .apk
  - x86 coming in the future

Getting and installing the NDK

- What's in the NDK?
  - Development tools
  - Stable native APIs system headers
  - **Documentation** - IMPORTANT
  - Samples
- Getting the NDK
  http://developer.android.com/sdk/ndk/index.html
- Prerequisites
  - Windows, Mac or Linux
  - Complete SDK
  - make (GNU's) and awk
  - For Windows, Cygwin 1.7 or higher
- NDK set up:
  - Make sure prerequisites are installed
  - Download and install NDK

Stable native APIs system headers

- libc (C library) headers
- libm (math library) headers
- JNI interface headers
- libz (Zlib compression) headers
- liblog (Android logging) header
- OpenGL ES 1.1 and OpenGL ES 2.0 (3D graphics libraries) headers
- libjnigraphics (Pixel buffer access) header (for Android 2.2 and above).
- Minimal set of headers for C++ support
- OpenSL ES native audio libraries
- Android native application APIs

Documentation I

- INSTALL.HTML - describes how to install the NDK and configure it for your host system
- OVERVIEW.HTML - provides an overview of the NDK capabilities and usage
- ANDROID-MK.HTML - describes the use of the Android.mk file, which defines the native sources you want to compile
- APPLICATION-MK.HTML - describes the use of the Application.mk file, which describes the native sources required by your Android application
- CPLUSPLUS-SUPPORT.HTML - describes the C++ support provided in the Android NDK
- CPU-ARCH-ABIS.HTML - a description of supported CPU architectures and how to target them.

Documentation II

- CPU-FEATURES.HTML - a description of the cpufeatures static library that lets your application code detect the target device's CPU family and the optional features at runtime.
- CPU-ARM-NEON.HTML - a description of how to build with optional ARM NEON / VFPv3-D32 instructions.
- CHANGES.HTML - a complete list of changes to the NDK across all releases.
- DEVELOPMENT.HTML - describes how to modify the NDK and generate release packages for it
- HOWTO.HTML - information about common tasks associated with NDK development
- IMPORT-MODULE.HTML - describes how to share and reuse modules

Documentation III

- LICENSES.HTML - information about the various open source licenses that govern the Android NDK
- NATIVE-ACTIVITY.HTML - describes how to implement native activities
- NDK-BUILD.HTML - describes the usage of the ndk-build script
- NDK-GDB.HTML - describes how to use the native code debugger
- PREBUILTS.HTML - information about how shared and static prebuilt libraries work
- STANDALONE-TOOLCHAIN.HTML - describes how to use Android NDK toolchain as a standalone compiler (still in beta).
- SYSTEM-ISSUES.HTML - known issues in the Android system images that you should be aware of, if you are developing using the NDK.
- STABLE-APIS.HTML - a complete list of the stable APIs exposed by headers in the NDK.

Sample applications I

- hello-jni - a simple application that loads a string from a native method implemented in a shared library and then displays it in the application UI.
- two-libs - a simple application that loads a shared library dynamically and calls a native method provided by the library. In this case, the method is implemented in a static library imported by the shared library.
- san-angeles - a simple application that renders 3D graphics through the native OpenGL ES APIs, while managing activity lifecycle with a GLSurfaceView object.
- hello-gl2 - a simple application that renders a triangle using OpenGL ES 2.0 vertex and fragment shaders.

Sample applications II

- hello-neon - a simple application that shows how to use the cpufeatures library to check CPU capabilities at runtime, then use NEON intrinsics if supported by the CPU. Specifically, the application implements two versions of a tiny benchmark for a FIR filter loop, a C version and a NEON-optimized version for devices that support it.
- bitmap-plasma - a simple application that demonstrates how to access the pixel buffers of Android Bitmap objects from native code, and uses this to generate an old-school "plasma" effect.
- native-activity - a simple application that demonstrates how to use the native-app-glue static library to create a native activity
- native-plasma - a version of bitmap-plasma implemented with a native activity.

Using the NDK

1. Place native code under `<project>/jni/...`
2. Create `<project>/jni/Android.mk` to describe native code to NDK
3. Optional: create `<project>/jni/Application.mk` for describing which natives sources are required by app
4. Build native code:

```
cd <project>
<ndk>/ndk-build
```

5. Compile app with SDK. Native code will be shared lib in .apk file.

Example native call in Activity

```
public class HelloJni extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        /* Create a TextView and set its content.
         * the text is retrieved by calling a native
         * function.
         */
        TextView  tv = new TextView(this);
        tv.setText( stringFromJNI() + " " + pid() );
        setContentView(tv);
    }

    /* A native method that is implemented by the
     * 'hello-jni' native library, which is packaged
     * with this application.
     */
    public native String  stringFromJNI();
...
    /* this is used to load the 'hello-jni' library on application
     * startup. The library has already been unpacked into
     * /data/data/com.example.HelloJni/lib/libhello-jni.so at
     * installation time by the package manager.
     */
    static {
        System.loadLibrary("hello-jni");
    }
}
```

Example native call implementation

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

Example Android.mk

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE    := hello-jni
LOCAL_SRC_FILES := hello-jni.c

include $(BUILD_SHARED_LIBRARY)
```

Implementing fully native applications

- Android 2.3 and up
- Native lifecycle management
- Still runs within context of dedicated Dalvik VM
- Can use JNI to call on Java functions

Limited APIs available to native apps

- Activity lifecycle management:
  `<android/native_activity.h>`
- Input events and sensors:
  `<android/looper.h>`
  `<android/input.h>`
  `<android/keycodes.h>`
  `<android/sensor.h>`
- Window management:
  `<android/rect.h>`
  `<android/window.h>`
  `<android/native_window.h>`
  `<android/native_window_jni.h>`
- Direct access to assets:
  `<android/configuration.h>`
  `<android/asset_manager.h>`
  `<android/storage_manager.h>`
  `<android/obb.h>`

Example fully native app Manifest

```
<?xml version="1.0" encoding="utf-8"?>
<!-- BEGIN_INCLUDE(manifest) -->
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.example.native_activity"
        android:versionCode="1"
        android:versionName="1.0">

    <!-- This is the platform API where NativeActivity was introduced. -->
    <uses-sdk android:minSdkVersion="8" />

    <!-- This .apk has no Java code itself, so set hasCode to false. -->
    <application android:label="@string/app_name" android:hasCode="false">

        <!-- Our activity is the built-in NativeActivity framework class.
             This will take care of integrating with our NDK code. -->
        <activity android:name="android.app.NativeActivity"
                android:label="@string/app_name"
                android:configChanges="orientation|keyboardHidden">
            <!-- Tell NativeActivity the name of or .so -->
            <meta-data android:name="android.app.lib_name"
                    android:value="native-activity" />
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Example fully native app

```
void android_main(struct android_app* state) {
    struct engine engine;

    // Make sure glue isn't stripped.
    app_dummy();

    memset(&engine, 0, sizeof(engine));
    state->userData = &engine;
    state->onAppCmd = engine_handle_cmd;
    state->onInputEvent = engine_handle_input;
    engine.app = state;

    // Prepare to monitor accelerometer
    engine.sensorManager = ASensorManager_getInstance();
    engine.accelerometerSensor = ASensorManager_getDefaultSensor(engine.sensorManager,
            ASENSOR_TYPE_ACCELEROMETER);
    engine.sensorEventQueue = ASensorManager_createEventQueue(engine.sensorManager,
            state->looper, LOOPER_ID_USER, NULL, NULL);

    if (state->savedState != NULL) {
        // We are starting with a previous saved state; restore from it.
        engine.state = *(struct saved_state*)state->savedState;
    }

    // loop waiting for stuff to do.
    while (1) {
...
    }
}
```

struct android_app I

```
/**
 * This is the interface for the standard glue code of a threaded
 * application.  In this model, the application's code is running
 * in its own thread separate from the main thread of the process.
 * It is not required that this thread be associated with the Java
 * VM, although it will need to be in order to make JNI calls any
 * Java objects.
 */
struct android_app {
    // The application can place a pointer to its own state object
    // here if it likes.
    void* userData;

    // Fill this in with the function to process main app commands (APP_CMD_*)
    void (*onAppCmd)(struct android_app* app, int32_t cmd);

    // Fill this in with the function to process input events.  At this point
    // the event has already been pre-dispatched, and it will be finished upon
    // return.  Return 1 if you have handled the event, 0 for any default
    // dispatching.
    int32_t (*onInputEvent)(struct android_app* app, AInputEvent* event);

    // The ANativeActivity object instance that this app is running in.
    ANativeActivity* activity;

    // The current configuration the app is running in.
    AConfiguration* config;
```

struct android_app II

```
// This is the last instance's saved state, as provided at creation time.
// It is NULL if there was no state.  You can use this as you need; the
// memory will remain around until you call android_app_exec_cmd() for
// APP_CMD_RESUME, at which point it will be freed and savedState set to NULL.
// These variables should only be changed when processing a APP_CMD_SAVE_STATE,
// at which point they will be initialized to NULL and you can malloc your
// state and place the information here.  In that case the memory will be
// freed for you later.
void* savedState;
size_t savedStateSize;

// The ALooper associated with the app's thread.
ALooper* looper;

// When non-NULL, this is the input queue from which the app will
// receive user input events.
AInputQueue* inputQueue;

// When non-NULL, this is the window surface that the app can draw in.
ANativeWindow* window;

// Current content rectangle of the window; this is the area where the
// window's content should be placed to be seen by the user.
ARect contentRect;
```

struct android_app III

```
    // Current state of the app's activity.  May be either APP_CMD_START,
    // APP_CMD_RESUME, APP_CMD_PAUSE, or APP_CMD_STOP; see below.
    int activityState;

    // This is non-zero when the application's NativeActivity is being
    // destroyed and waiting for the app thread to complete.
    int destroyRequested;

    // -------------------------------------------------
    // Below are "private" implementation of the glue code.

    pthread_mutex_t mutex;
    pthread_cond_t cond;

    int msgread;
    int msgwrite;

    pthread_t thread;

    struct android_poll_source cmdPollSource;
    struct android_poll_source inputPollSource;

    int running;
    int stateSaved;
    int destroyed;
    int redrawNeeded;
    AInputQueue* pendingInputQueue;
    ANativeWindow* pendingWindow;
    ARect pendingContentRect;
};
```