Cellular Networks and Mobile Computing
COMS 6998-11, Fall 2012

Instructor: Li Erran Li
(lierranli@cs.columbia.edu)

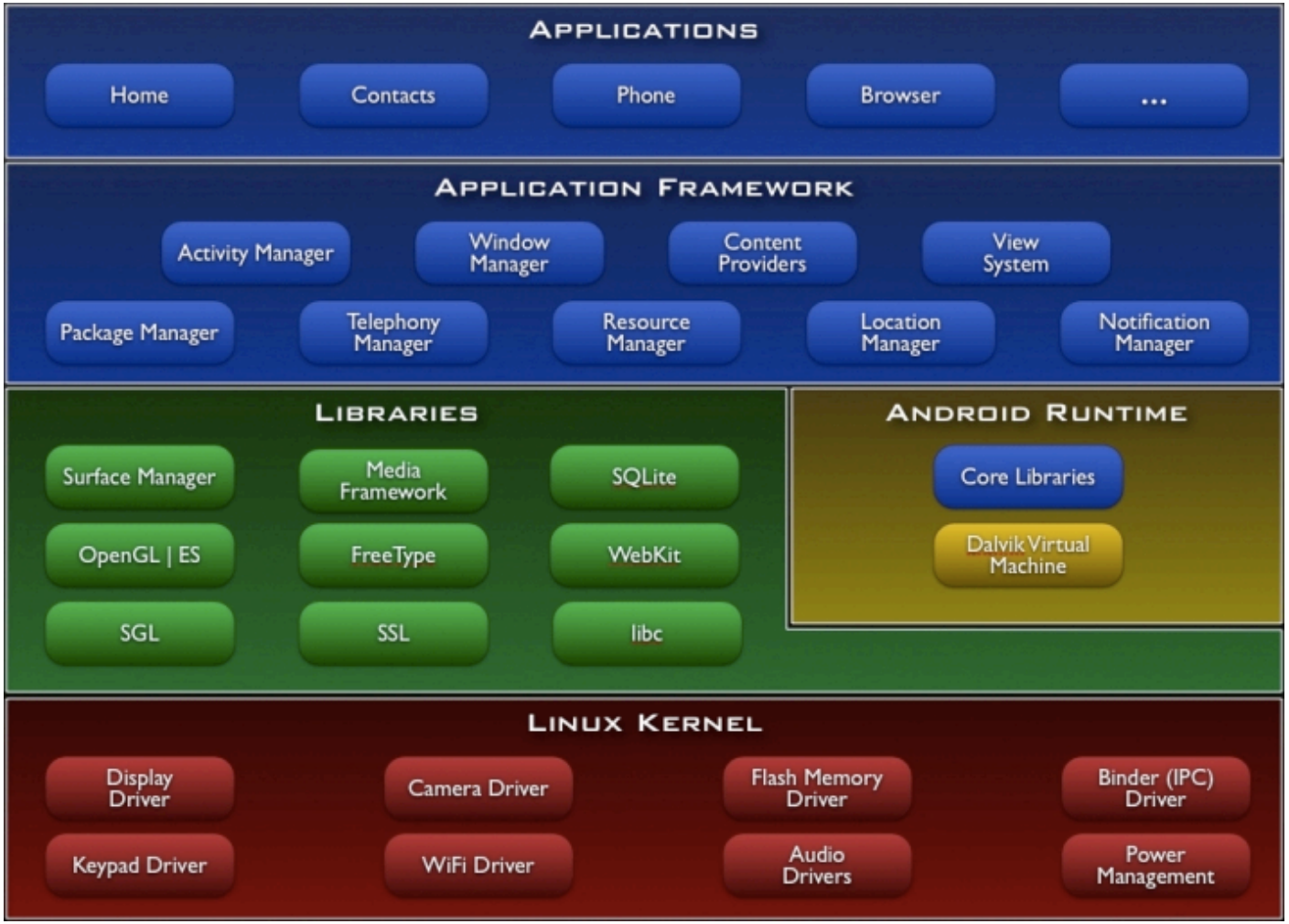
http://www.cs.columbia.edu/~lierranli/coms6998-11Fall2012/

9/18/2012: Introduction to Android

Outline

- Android OS Overview
- Android Development Process
- Eclipse and Android SDK Demo
- Application Framework
  - Activity, content provider, broadcast receiver, intent
- Android App Framework Demo
- Networking
- Google Cloud Messaging (GCM)

Android Architecture

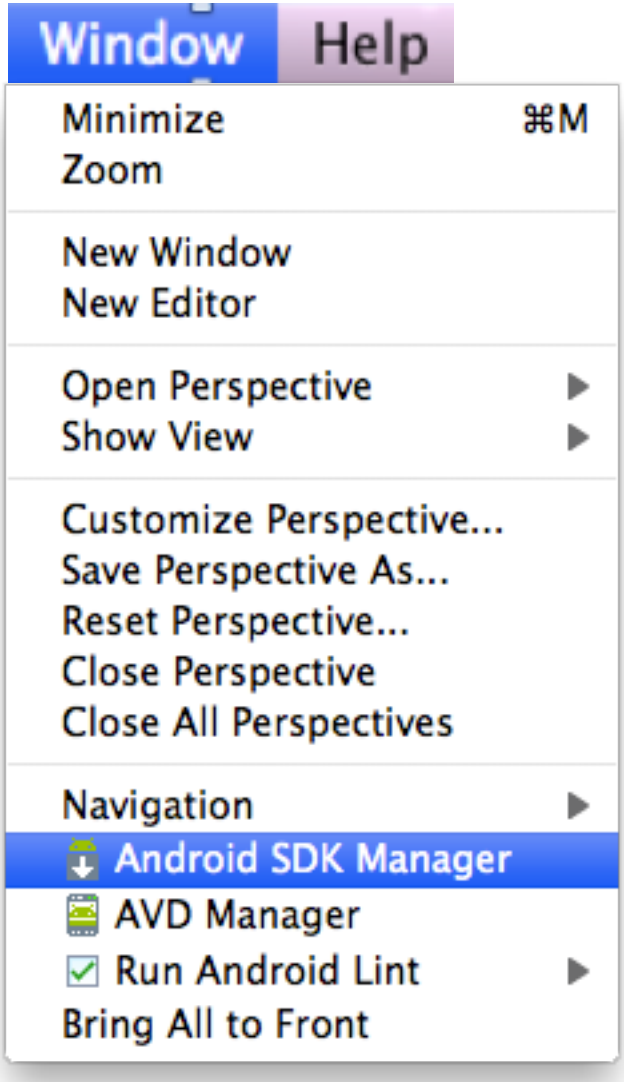


Android Development Process

- Setup develop environment (SDK, Eclipse)
  - SDK: compiler, debugger, device emulator
    - Multiplatform support: Windows, Mac, Linux
  - Java programming: has its own Java Virtual Machine and special byte code
- Create app
  - Android project containing java files and resource files
- Test app
  - Pack project into debuggable *.apk
  - Install, run and debug on emulator or device
- Publish app in Android market
- Get rich!

Setup SDK with Eclipse

- Download and install
  - Java Development Kit (JDK)
  - Eclipse
- Install and configure Android SDK plugin in Eclipse
  - Install Android Development Tools (ADT) plugin
    - Follow instructions on http://developer.android.com/sdk/installing/installing-adt.html
  - Eclipse will prompt you to specify Android SDK directory
  - Use Android SDK manager to install specific versions of Android



Android SDK Manager

Android SDK Manager

SDK Path: /Users/erranlli/erranlli-programming/android-sdk-macosx/

Packages

Name	API	Rev.	Status
Tools			
Android SDK Tools		20	Update available: rev. 20.0.3
Android SDK Platform-tools		12	Update available: rev. 14
Android 4.1 (API 16)			
Documentation for Android SDK	*16*	*2*	*Not installed*
SDK Platform	*16*	*2*	*Not installed*
Samples for SDK	*16*	*1*	*Not installed*
ARM EABI v7a System Image	*16*	*2*	*Not installed*
Intel x86 Atom System Image	*16*	*1*	*Not installed*
Mips System Image	*16*	*1*	*Not installed*
Google APIs	*16*	*2*	*Not installed*
Sources for Android SDK	*16*	*2*	*Not installed*
Android 4.0.3 (API 15)			
Documentation for Android SDK	15	1	Installed
SDK Platform	15	2	Update available: rev. 3
Samples for SDK	15	1	Update available: rev. 2
ARM EABI v7a System Image	15	1	Update available: rev. 2

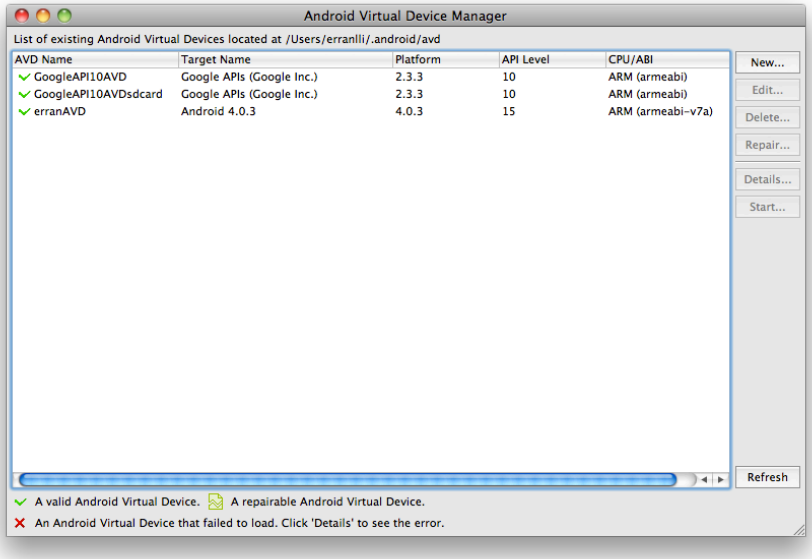
Show: Updates/New Installed Obsolete Select New or Updates Install 15 packages...

Sort by: API level Repository Deselect All Delete 7 packages...

Done loading packages.

Option 1: Use an Emulator

- Create an Android Virtual Device (AVD)
  - Lets you specify the configuration of a device to be emulated by the Android Emulator
  - Create AVD in Eclipse by selecting Window>AVD Manager

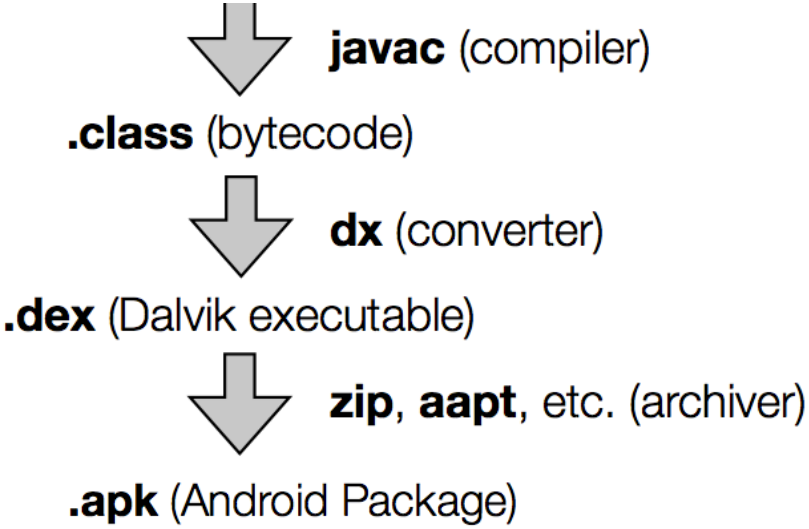


Option 2: Use a Device

- Install drivers for device
- Connect device to a computer via USB cable
  - Make sure turned on USB debugging on device (Settings -> Application -> Development -> USB debugging)
- Device will be recognized within Eclipse (DDMS view)

Android Application Framework

- Runs in its own virtual machine & process
  - Isolation among apps
- Is composed of basic components
- App components can be activated when any of its components needs to be executed

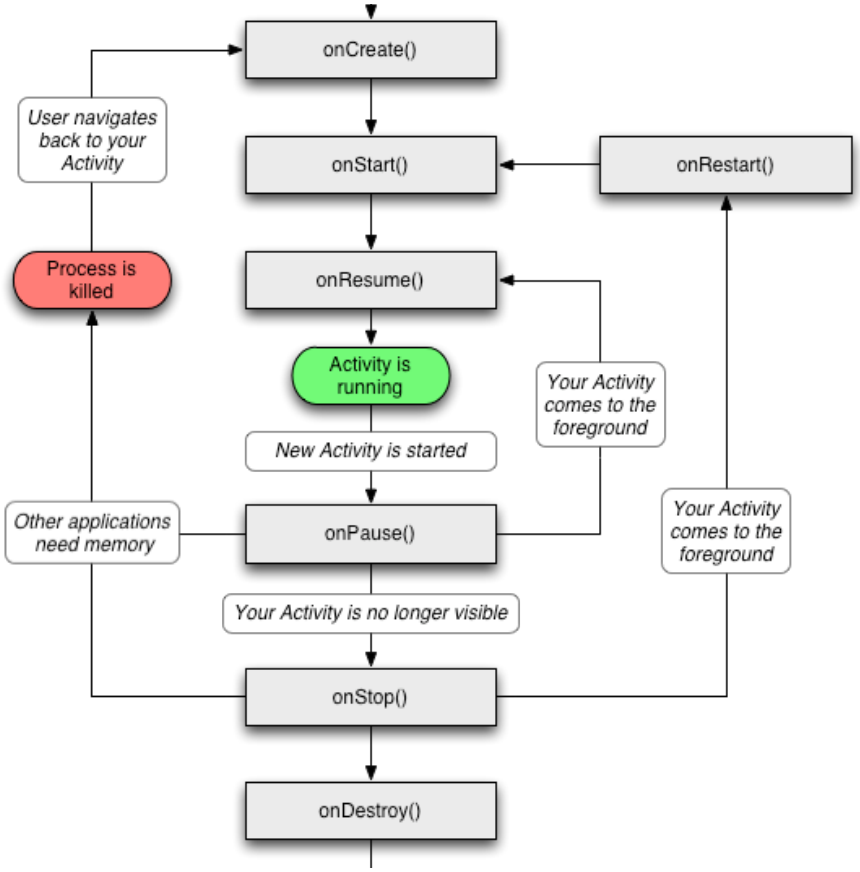


Android App Components

Basic Components	Description
Activity	Deals with UI aspects. Typically corresponds to a single screen
Service	Background tasks (e.g. play music in background while user is web surfing) that typically have no UI
BroadCastReceiver	Can receive messages (e.g. “low battery”) from system/apps and act upon them
ContentProvider	Provide an interface to app data. Lets apps share data with each other.

Activity

- UI portion of an app
- One activity typically corresponds to a single screen of an app
- Conceptually laid out as a stack
  - Activity on top of stack visible in foreground
  - Background activities are stopped but state is retained
  - Back button resumes previous Activity in stack
  - HOME button moves app and its activity in background



Activity Example

```
MainActivity.java
public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
         // savedInstanceState holds any data that may have been saved
         // for the activity before it got killed by the system (e.g.
         // to save memory) the last time

        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
}
```

```
AndroidManifest.xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"   package="com.example.hellotest" >
  <application
    android:label="@string/app_name" >
    <activity
      android:name=".MainActivity" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

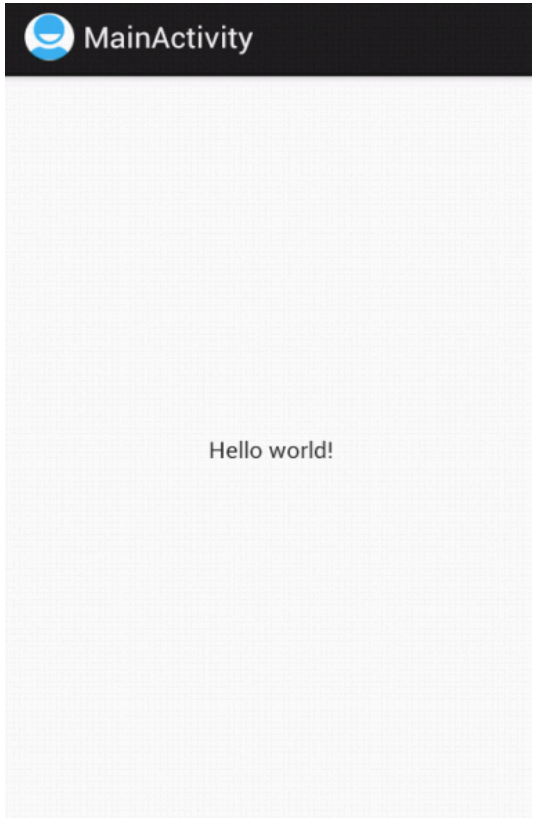
Views

- Views are building blocks of UI
  - TextView, ListView, MapView, ImageView …

```
Main.xml
<?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:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />
</LinearLayout>
```

```
MainActivity.java
public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
            // savedInstanceState holds any data that may have been saved
            // for the activity before it got killed by the system (e.g.
            // to save memory) the last time

        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
}
```

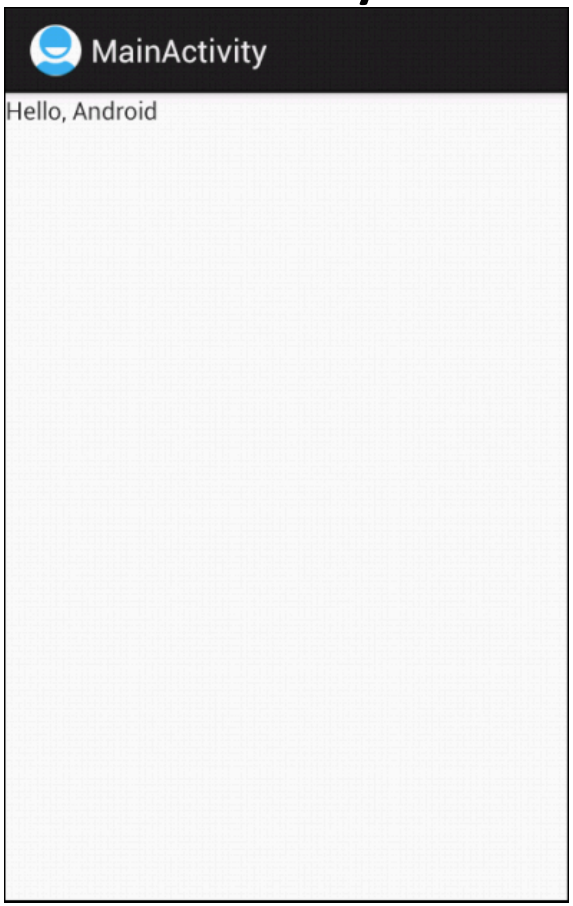


Views (Cont'd)

- Views can also be created programmatically

```
MainActivity.java
public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
            // savedInstanceState holds any data that may have been saved
            // for the activity before it got killed by the system (e.g.
            // to save memory) the last time

        super.onCreate(savedInstanceState);
         //setContentView(R.layout.main);
        TextView tv = new TextView(this);
        tv.setText("Hello, Android");
        setContentView(tv);
}
```



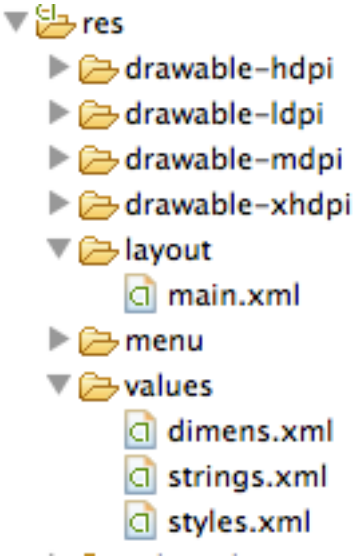
Layouts

- Controls how Views are laid out: LinearLayout, TableLayout, RelativeLayout

```
Main.xml
<?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:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />
</LinearLayout>
```

```
MainActivity.java
public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
            // savedInstanceState holds any data that may have been saved
            // for the activity before it got killed by the system (e.g.
            // to save memory) the last time

        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
}
```



Resources

- Reference included content via R.java

`res/anim/`	XML files for frame-by-frame animation
`res/drawable/`	images compiled and optimized
`res/layout/`	XML files for screen layouts
`res/values/`	compiled XML files into different resource
`res/xml/`	arbitrary XML files
`res/raw/`	raw, uncompiled files

Services

- Faceless components that typically run in the background
  - Music player, network download, etc
- Services can be started in two ways
  - A component can start the service by calling startService()
  - A component can call bindService() to create the service
- Service started using startService() remains running until explicitly killed
- Service started using bindService() runs as long s the component that created it is still “bound” to it.
- The Android system can force-stop a service when memory is low
  - However “foreground” services are almost never killed
  - If the system kills a service, it will restart the service as soon as resource is available

Services Example

BackgroundSoundServie.java

```
public class BackgroundSoundService extends Service {
            MediaPlayer player;
            public void onCreate() {
                        super.onCreate();
                        player = MediaPlayer.create(this, R.raw.waltz);
                        player.setLooping(false);
                        player.setVolume(100,100);
            }
            public int onStartCommand(Intent intent, int flags, int startId) {
                        player.start();
                         return 1;
    }
}
```

AndroidManifest.xml

```
 <service
          android:enabled="true"
          android:name=".BackgroundSoundService" />
```

MainActivity.java

```
 public class MainActivity extends Activity {
     @Override
     public void onCreate(Bundle savedInstanceState) {
         …
         Intent svc=new Intent(this, BackgroundSoundService.class);
         startService(svc);
     }
 }
```

Broadcast Receivers

- Components designed to respond to broadcast messages (called Intents)
- Can receive broadcast messages from the system. For example when:
  - A new phone call comes in
  - There is a change in the battery level or cellID
- Can receive messages broadcast by apps
  - Apps can also define new broadcast messages

Broadcast Receivers Example

- Listen to phone state changes

```
AndroidManifest.xml
<uses-permission android:name="android.permission.READ_PHONE_STATE" >
<receiver android:name="MyPhoneReceiver" >
        <intent-filter>
          <action android:name="android.intent.action.PHONE_STATE" >
          </action>
        </intent-filter>
      </receiver>
```

```
public class MyPhoneReceiver extends BroadcastReceiver {
                public void onReceive(Context context, Intent intent) {
                Bundle extras = intent.getExtras();
                  if (extras != null) {
                    String state = extras.getString(TelephonyManager.EXTRA_STATE);
                    if (state.equals(TelephonyManager.EXTRA_STATE_RINGING)) {
                      String phoneNumber = extras.getString(TelephonyManager.EXTRA_INCOMING_NUMBER);
                      Toast.makeText(context, "Incoming number: "+phoneNumber,
                            Toast.LENGTH_LONG).show();
                    }
                  }
                }
}
```

Content Providers

- Enable sharing of data across apps
  - Address book, photo gallery, etc.
- Provides uniform APIs for
  - Query, delete, update, and insert rows
  - Content is represented by URI and MIME type
- API: extends ContentProvider implement methods such as insert, delete, query, update, oncreate

Content Providers Example

AndroidManifest.xml

```
<uses-permission
android:name="android.permission.READ_CONTACTS"/>
```

MainActivity.java

```
 public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        …
        Cursor people = getContentResolver().query(ContactsContract.Contacts.
                                CONTENT_URI, null, null, null, null);

        while(people.moveToNext()) {
           int nameFieldColumnIndex = people.getColumnIndex
                    (PhoneLookup.DISPLAY_NAME);
           String contact = people.getString(nameFieldColumnIndex);
           contactView.append("Name: ");
           contactView.append(contact);
           contactView.append("\n");
        }
        people.close();
     }
}
```

Intent

- Intent are messages used for activating components
- Intent object
  - Help identify the receiving components
  - May contain action to be take and data to act on
  - Serve as notification for a system event (e.g. new call)
- Intents can be
  - Explicit: specify receiving component (java class)
  - Implicit: specify action/data. Components registered for the action/data pair can receive the Intent
    - Register via IntentFilters in AndroidManifest.xml
    - BroadCastReceivers can also register programmatically

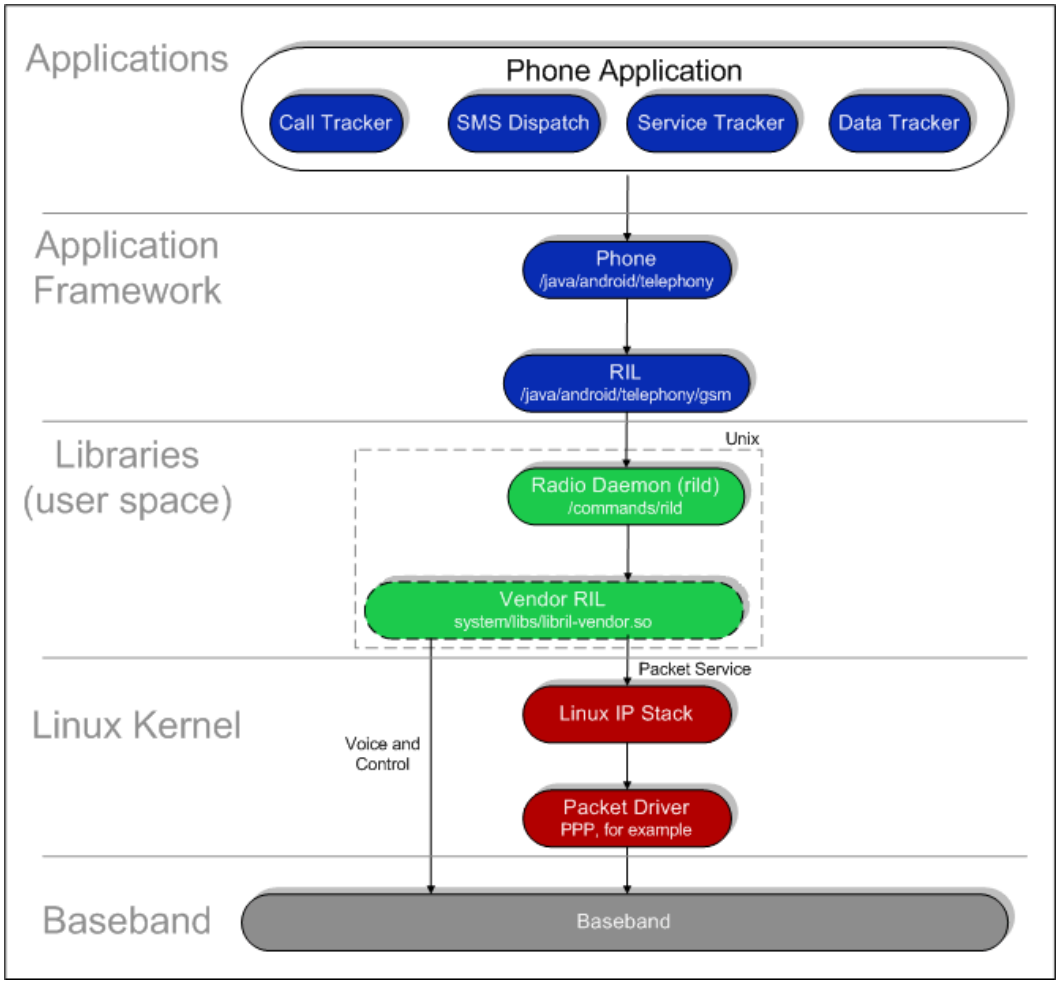
Networking

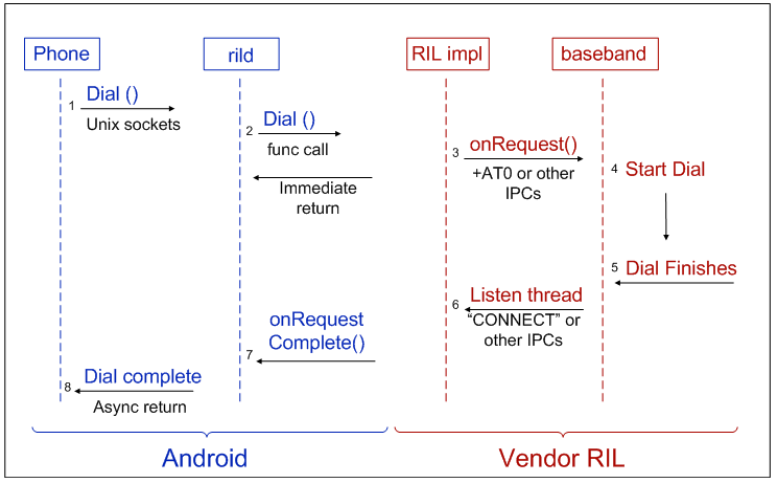
- Net APIs
  - Standard java networking APIs
  - Two Http clients: HttpURLConnection and Apache Http client

Telephony APIs (android.telephony)

- Send and receive SMS
- Get mobile network info (network type, operator, …)
- Get current value of network parameters (cellID, signal strength, SNR, roaming state …)
- Monitor state changes (cellID, call state, connectivity …)
- Get current device state (connected, idle, active)
- Get device paramesters (IMSI, IMEI, device type)

Android Telephony Deep Dive

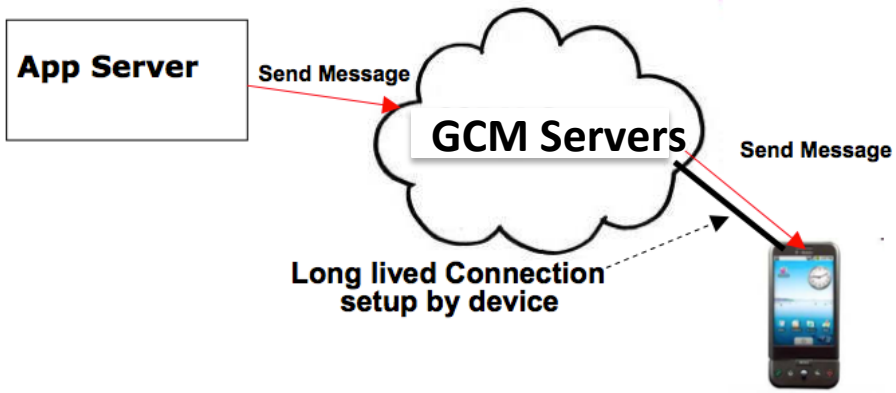




Google Cloud Messaging (GCM)

- Various mechanisms to keep an app in synch with changes in the server (e.g. twitter, facebook)
  - Polling: app periodically polls the servers for changes
  - Push: servers push changes to app
- Polling can be inefficient if server data changes infrequently
  - Unnecessary battery drain and network overhead (signaling and data)
- Several apps polling independently without coordination can also be inefficient
  - High battery drain and radio signaling every time the devices moves from IDLE to CONNECTED state

Google Cloud Messaging (Cont'd)



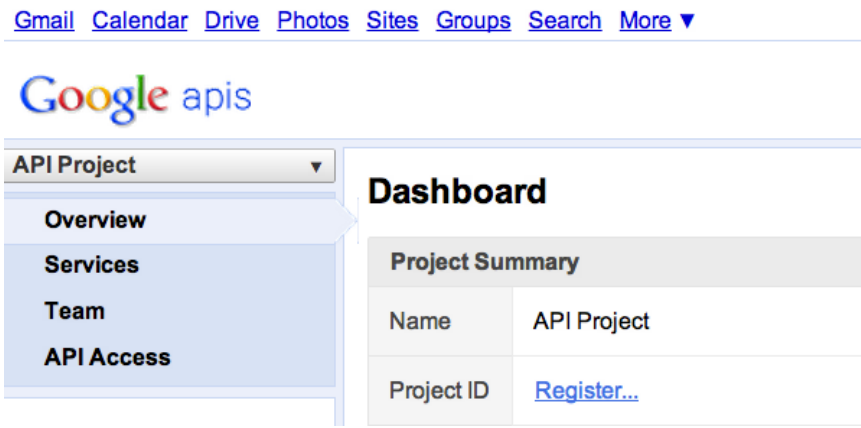
- Push notification problems
  - Network firewalls prevent servers from directly sending messages to mobile devices
- GCM solution
  - Maintain a connection between device and Google GCM server
  - Push server updates to apps on the device via this connection
  - Optimize this connection to minimize bandwidth and battery consumption (e.g. adjusting the frequency of keep alive messages)

Google Cloud Messaging (Cont'd)

C2DM is deprecated, accepts no new users

Step 1

- Create a Google API project from Google APIs console page https://code.google.com/apis/console/#project:908058729336
  - Enable GCM service
  - Obtain an API key
  - Create new server key
  - Install helper libraries



Google Cloud Messaging (Cont'd)

Step 2

- Write the Android app
  - Copy gcm.jar file into your app classpath
  - Configure manifest file for SDK version, permission
  - Add broadcast receiver
  - Add intent service
  - Write my_app_package.GCMIntentService class
  - Write main activity

```
import com.google.android.gcm.GCMRegistrar;
…
GCMRegistrar.checkDevice(this);
    GCMRegistrar.checkManifest(this);
    final String regId = GCMRegistrar.getRegistrationId(this);
    if (regId.equals("")) {
      GCMRegistrar.register(this, SENDER_ID);
    } else {
      Log.v(TAG, "Already registered");
    }
```

Google Cloud Messaging (Cont'd)

Step 3

- Write server-side app
  - Copy gcm-server.jar file from the SDK's gcm-server/dist directory to your server class path
  - Create a servlet that can be used to receive client's GCM registration ID
  - Create a servlet to unregister registration ID
  - Use com.google.android.gcm.server.Sender helper class from GCM library to send a message to client

```
import com.google.android.gcm.server.*;

Sender sender = new Sender(myApiKey);
Message message = new Message.Builder
().build();
MulticastResult result = sender.send
(message, devices, 5);
```

Online Resources

- Android API: http://developer.android.com/reference/packages.html
- Basics http://developer.android.com/guide/components/index.html
- GCM: http://developer.android.com/guide/google/gcm/index.html

Questions?