CS371m - Mobile Computing

Services and Broadcast Receivers

Clicker

- Is it possible for your app to accomplish work when the forefront Activity is **not** one from your app?

A. Yes

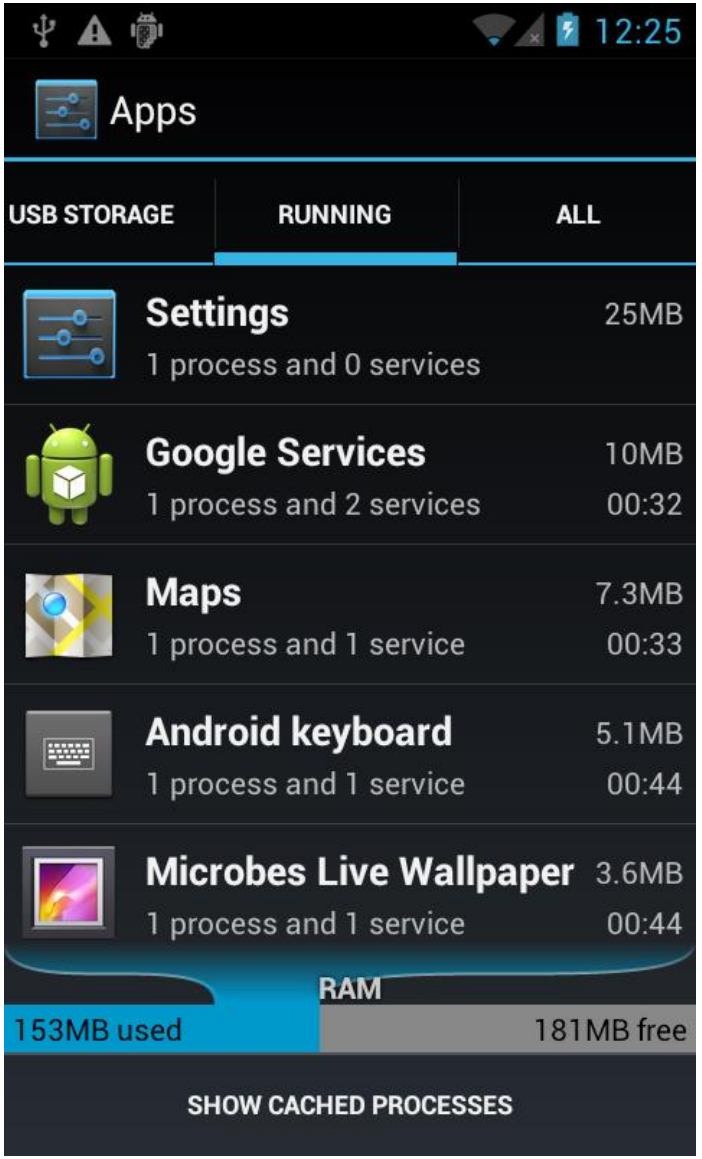
B. No

Services

- One of the four primary application components:
  - activities
  - content providers
  - services
  - broadcast receivers

Services - Purpose

- Application component that performs long-running operations in background with no UI
- application starts service and service continues to run even if original application ended or user moves to another application
- a way to run code when one of app's activity is not the forefront activity



Service - Examples

- Download app from store
  - leave store app, but download continues
  - any kind of download of upload via network
- Play music even when music player dismissed (if user desires)
- maintain connection in chat app when phone call received
- periodically poll website for updates / changes
  - the grade checker
  - may lead to notification

Clicker

- Will a class in your app that extends the Service class have an xml layout file?

A. Yes

B. No

Clicker

- Do services need to be declared in the app manifest file like activities?

A. Yes

B. No

Service Basics

- No User Interface Components
- Belongs to an app
  - must be declared in the app manifest
- May be running even if app is not at forefront, interacting with the user
- May be private (usable only by the app they belong to) or public (usable by apps other than yours)

Starting Services

- Two ways to start services:

1. Manually by an app using a call to the API - recall the `startActivity` method for Activities
2. Another activity tries to connect (bind) to a service via inter process communication - may be an app other than yours if your make your service public

Stopping Services

- Services will run until shut down:

1. by themselves when task completed or possibly by owning app
   `stopSelf`
2. By the owning app via a call to
   `stopService`
3. If Android needs the RAM the service is using
   - just like it does for activities deep in the activity stack

Types Services - **Started** or Bound

- Started:
  - application component, such as an Activity, starts the service with the method call **startService()**
  - once started service can run in background indefinitely
    - clean up after yourself
  - generally services do not return a result (see bound service)
    - can send a local broadcast when done if necessary to notify Activity or other Service
  - service should stop itself when done
- **Most services in our projects are *started***

Forms of Services - Started or **Bound**

- Bound
  - application component binds itself to existing service via the bindService() method
  - bound service provides client-server interface that allows application component to interact with service
  - interact with service, send requests, get result via IPC (inter process communication)
  - service runs as long as one or more applications bound to it
  - destroyed when no applications bound

Forms of Services

- Service can be started and later bound to other applications
  - so, both started and bound
- private service (manifest) cannot be bound by other applications

Communicating with Services

- Two ways:
- Commands
  - no lasting connection to service
  - example: start service, end service
- Binding
  - creates communication channel between the service and other component
  - other component typically an activity or perhaps another service

Service or Thread

- Past examples, kept UI thread responsive with other threads of execution, especially AsyncTask
- Should services be used for this?
- Service for actions that need to take place even if user not interacting with UI or has closed application
- Example, do complex rendering of image to display to user.
  - Not a job for a service

Service Setup

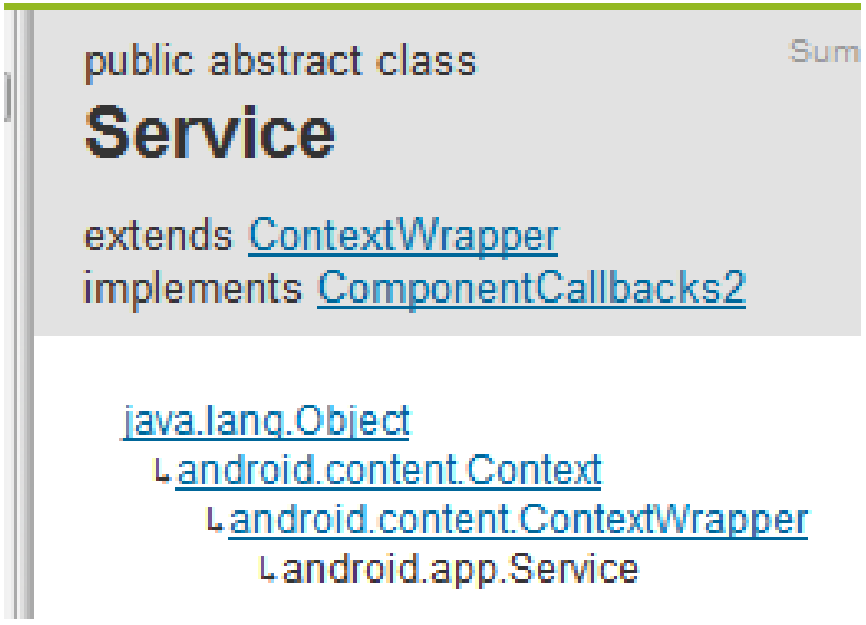
- Must declare Services in manifest
  - just like activities
  - otherwise when Service started, app will crash

```
</activity>

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

No intent filter for Service makes it private.

Creating a Service Class

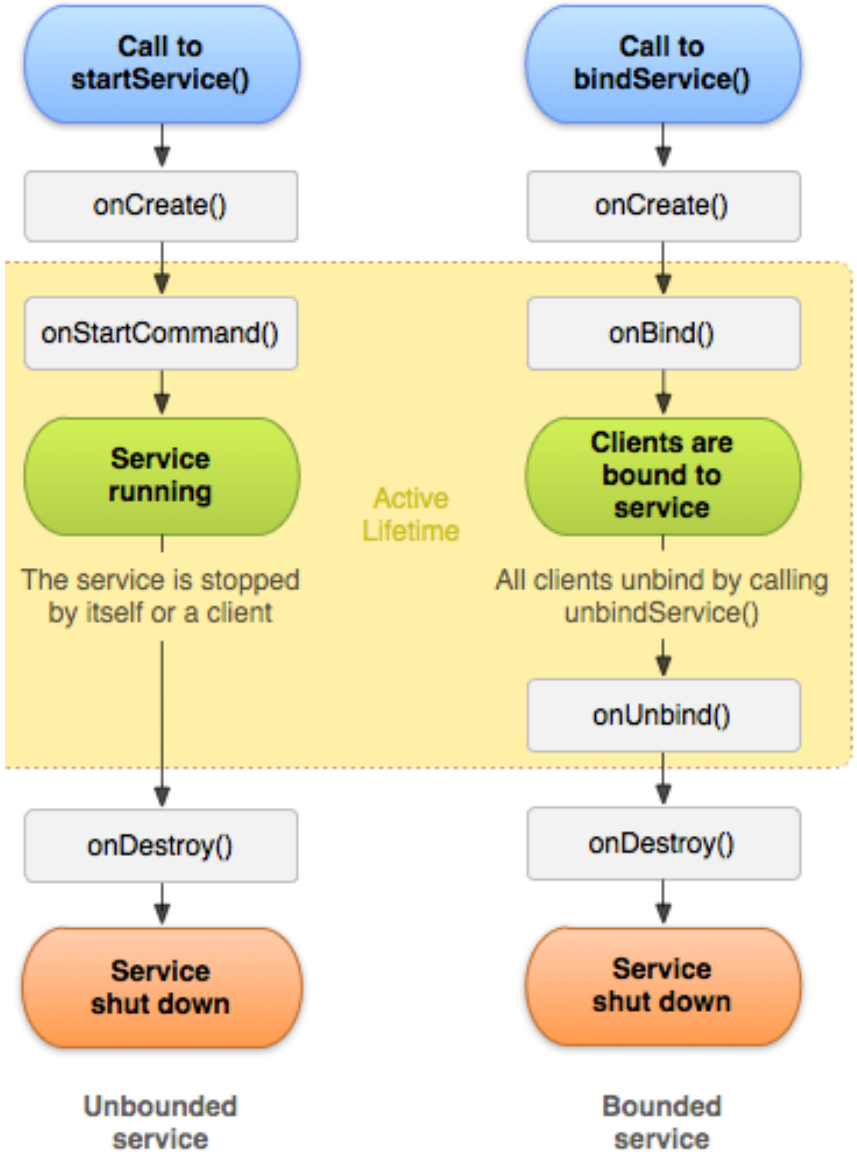
- create subclass of Android Service class or one of its existing subclasses
  - commonly IntentService
- override callback methods that handle important aspects of service lifecycle
- most important of these are:
  - onStartCommand
  - onBind (bound services)
  - onCreate
  - onDestroy
  - stopSelf



Service Lifecycle

- If component starts service with **startService** method (leads to call to **onStartCommand**) service runs until it calls **stopSelf** or another activity calls **stopService**
- if component calls **bindService** (**onStartCommand** not called) service runs as long as at least one component bound to it

Service Lifecycle



Service Responsiveness

- Services run on the main thread of hosting process
- By default a Service does **not** create a separate thread of execution
- If plan to do intensive CPU ops or blocking ops within Service, must still create a separate thread of execution to avoid ANRs
- **IntentService (subclass of Service) uses a worker thread to handle start requests**

APP EXAMPLE THAT USES A SERVICE

Service App Example

- From Roger Wallace
  - Spring 2011
  - wanted an app that would respond to texts (SMS) received when driving and respond with a message ("Driving - Get back to you soon.")
  - Initial version simply auto responds to all texts
  - how to change it so it responds only when driving?

Interesting Sidetrack - Disallowed?

- Google Play Developer Policy

Message Spam

We don't allow apps that send SMS, email, or other messages on behalf of the user without giving the user the ability to confirm the content and intended recipients.

Safe Driving Text Machine

Warez My Software Communication 914

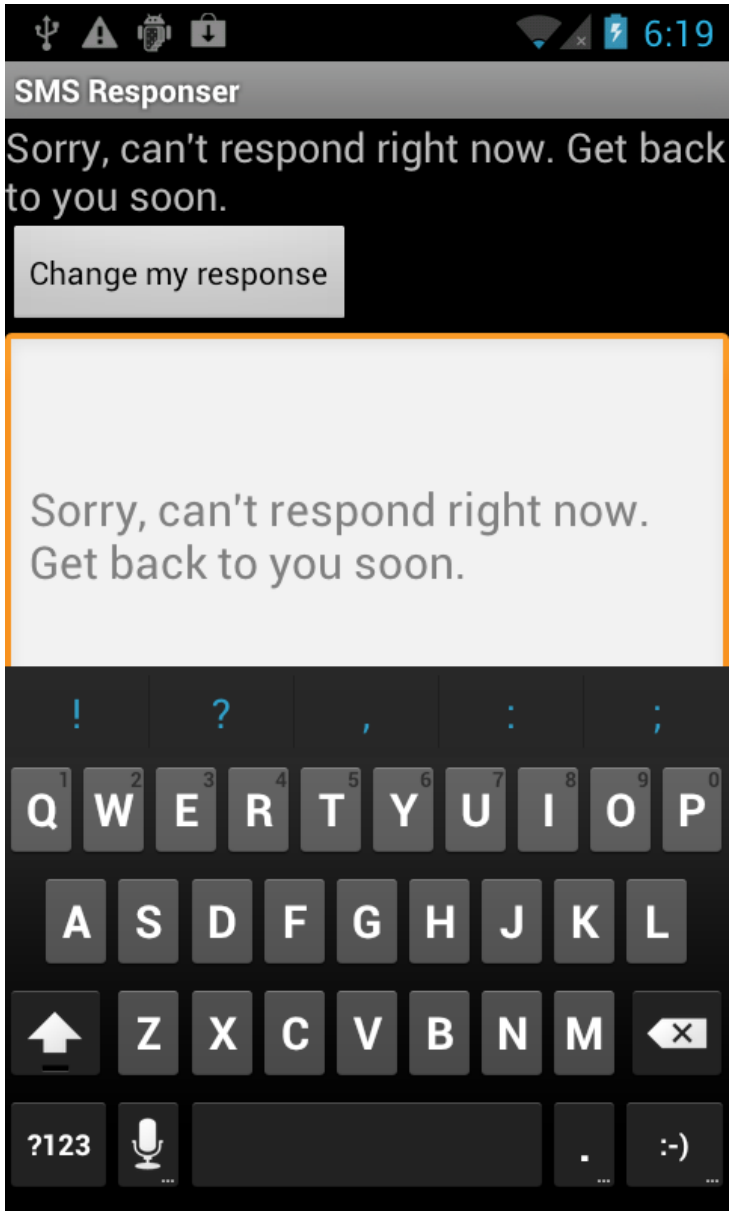
Everyone

You don't have any devices

Add to Wishlist Install

Example Service Application

- From *The Android Developer's Cookbook*
- SMSResponder Application
- Response stored in shared preferences
- App allows changes to message, start auto SMS responses and stop auto SMS respones



Using SMS

- Permission in manifest file to send and / or receive SMS messages

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

SMSResponder Basic App

- onCreate
- set up layout

```
@Override
public void onCreate(Bundle savedInstanceState) {
    Log.v(TAG,"In onCreate method");
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    myPrefs
        = PreferenceManager.getDefaultSharedPreferences(this);
    tv1 = (TextView) this.findViewById(R.id.display);
    ed1 = (EditText) this.findViewById(R.id.editText);
}
```

SMSResponder onResume

```
public void onResume() {
    super.onResume();
    Log.v(TAG, "in onResume");
    reply = myPrefs.getString("reply", "Thank you " +
            "for your message. " +
            "I am busy now. I will call you later");
    tv1.setText(reply);
    updater = myPrefs.edit();
    ed1.setHint(reply);
    // startMyService();
}
```

Change Auto Response Message

```
public void changeAutoReplyText(View v) {
    Log.v(TAG, "in change respond text");
    reply = ed1.getText().toString();
    if(reply.length() == 0)
        reply = "Thank you for your message. " +
                "I am busy now. " +
                "I will call you later.";
    tv1.setText(reply);
    SharedPreferences.Editor updater = myPrefs.edit();
    updater.putString("reply", reply);
    updater.apply();
}
```

Starting Service

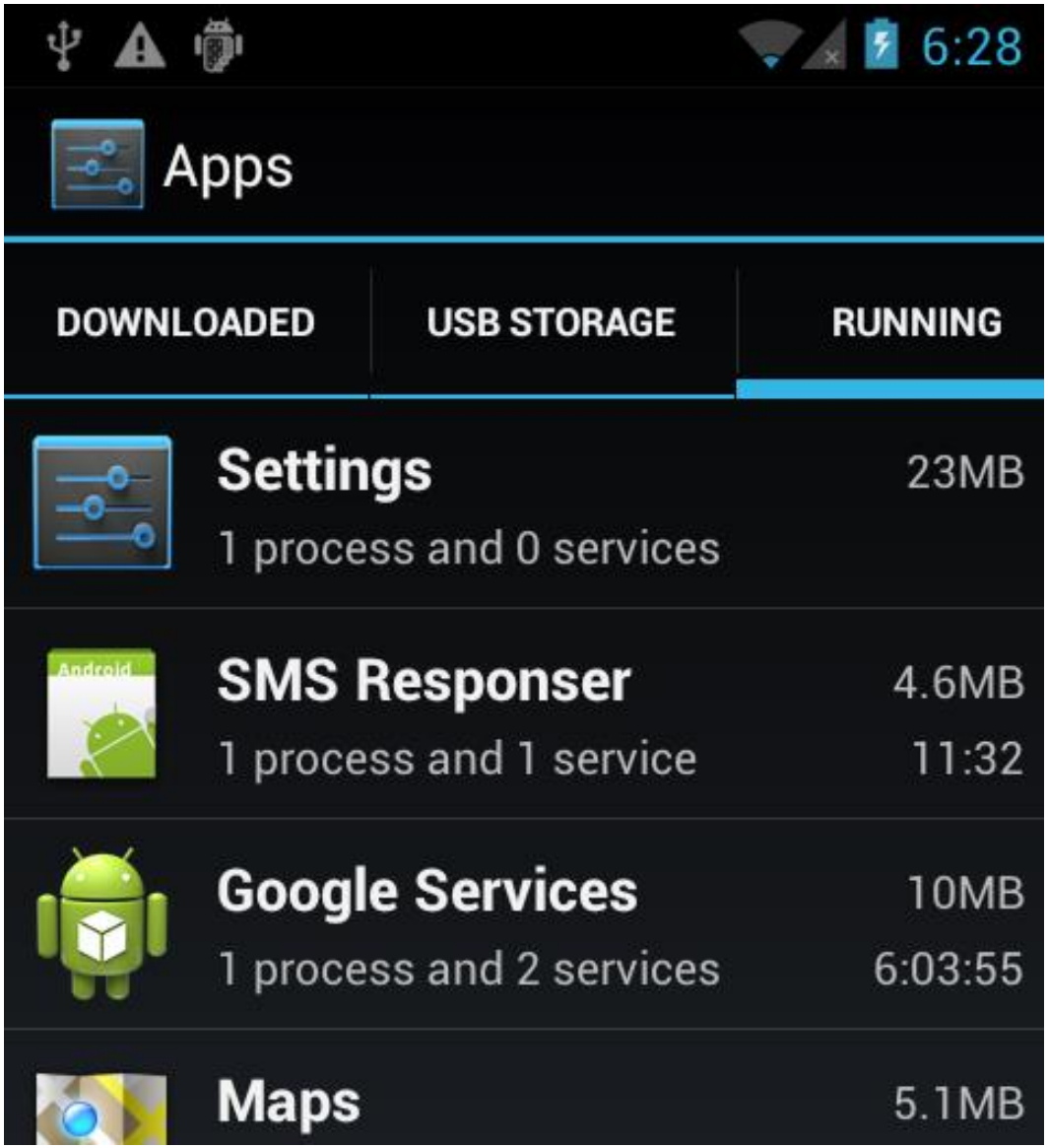
```
public void startAutoRespond(View v) {
    Log.v(TAG, "in start auto respond ");
    startMyService();
}


private void startMyService() {
    Log.v(TAG,"In startMyService method");
    boolean running = isMyServiceRunning();
    Log.d(TAG, "running: " + running);
    if(!running) {
        try {
            // start Service
            Intent svc = new Intent(this, ResponserService.class);
            startService(svc);
        }
        catch (Exception e) {
            Log.e("onCreate", "service creation problem", e);
        }
    }
}
```

Check if Service Already Running

```
private boolean isMyServiceRunning() {
    Log.v(TAG, "checking if service is running");
    ActivityManager manager
        = (ActivityManager) getSystemService(ACTIVITY_SERVICE);
    boolean result = false;
    // log all services to demo in class
    for (RunningServiceInfo service
                : manager.getRunningServices(Integer.MAX_VALUE)) {
        Log.v(TAG, "Running service: " + service.service.getClassNam
        if (serviceName.equals(service.service.getClassName())) {
            result = true;
        }
    }
    return result;
}
```

Alternative: Use public, static variable or method in Service class to indicate if running or not.

Service Running

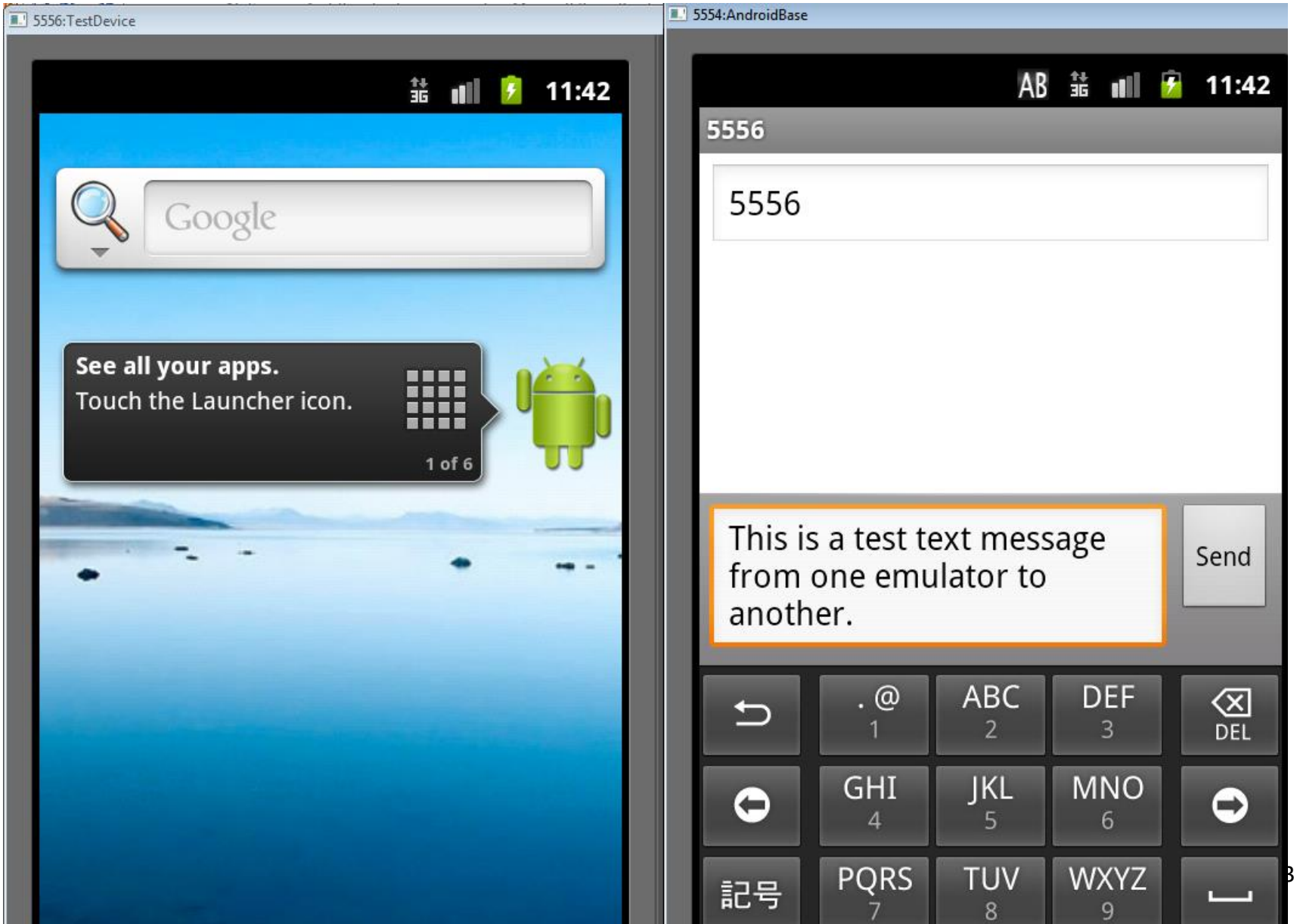


Service is running

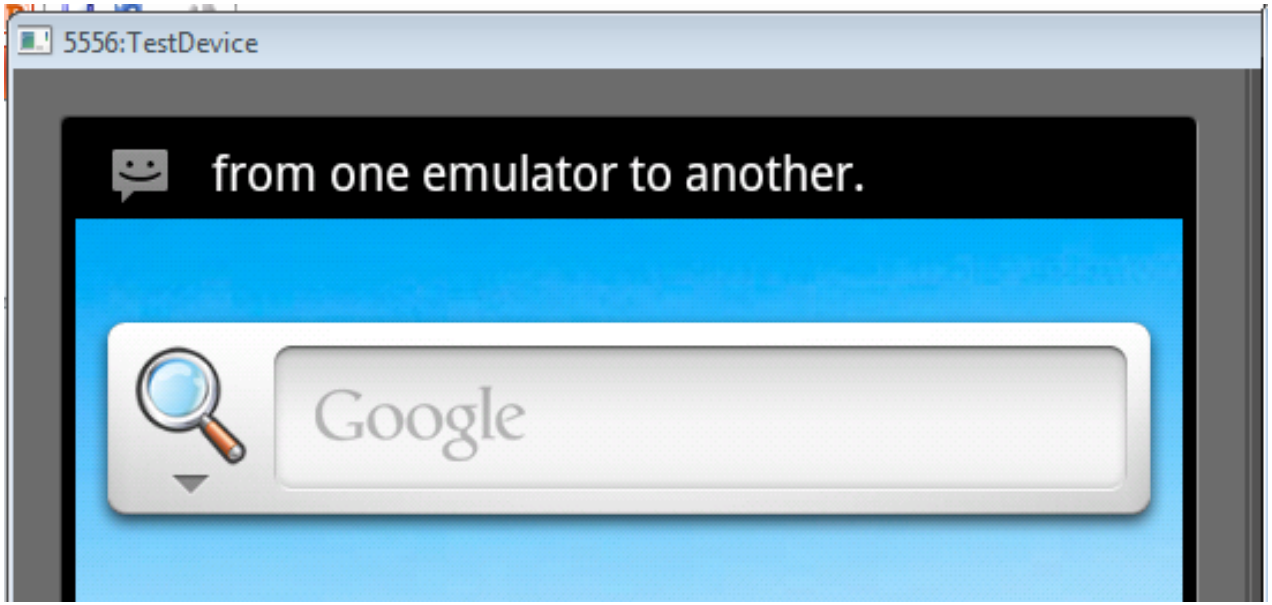
Simulating Texts

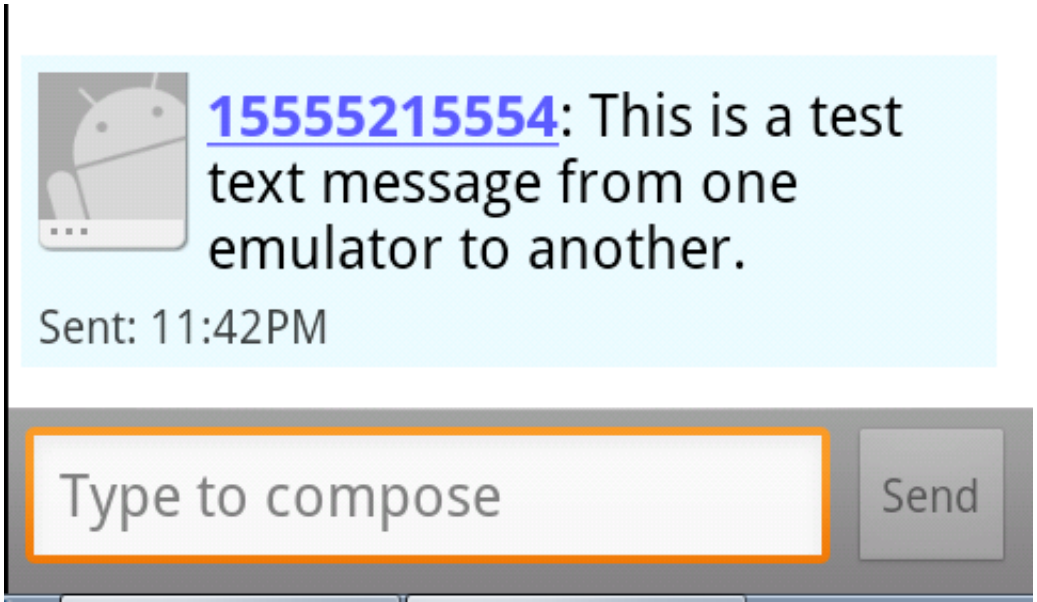
- Calls and texts can be simulated between emulators
- Start two emulators
- Use messaging app to send text
- Phone number is simply the emulator port number (visible at top of the emulator or in eclipse)

Dual Emulators

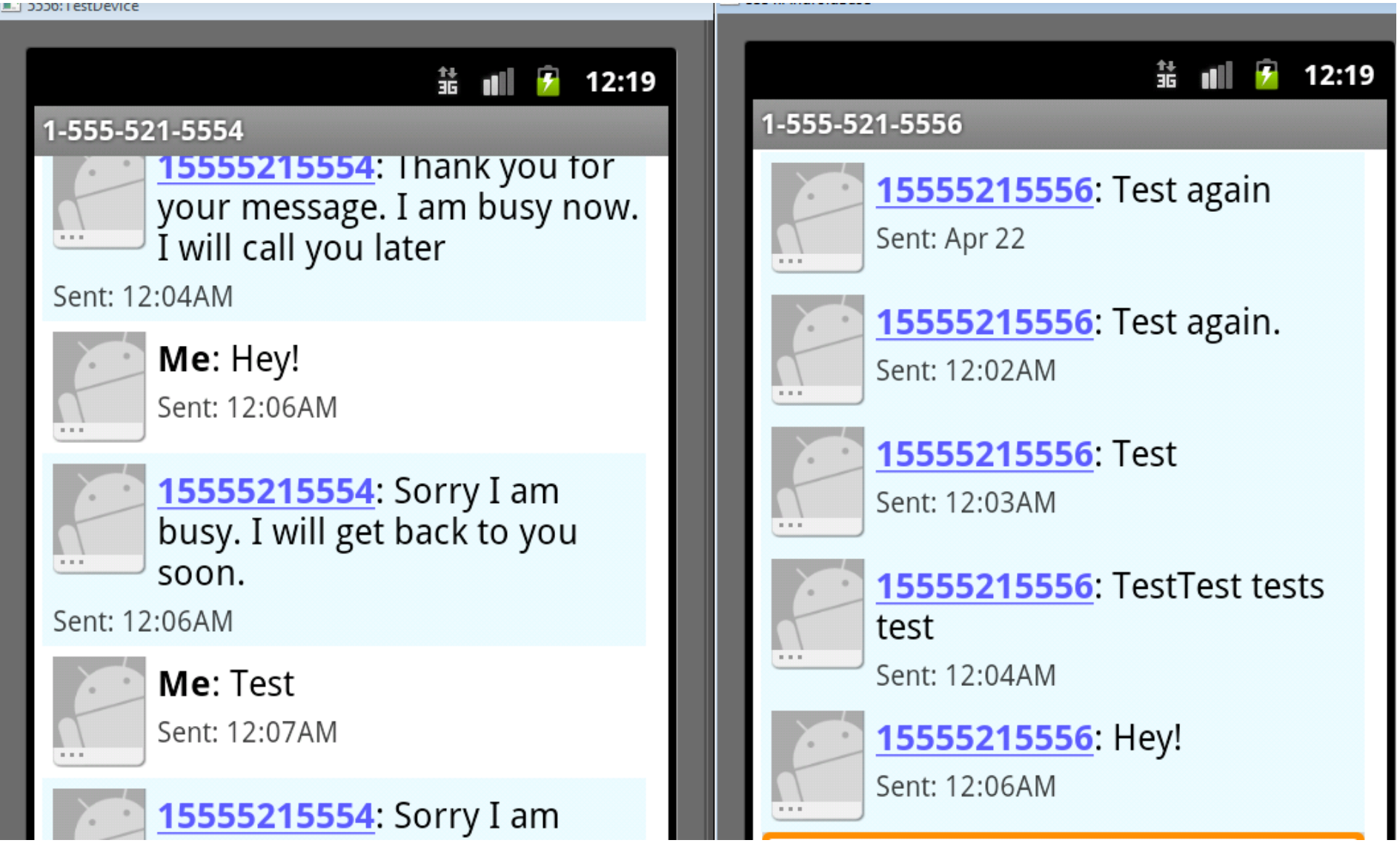


Emulator Texts





Testing Service



Creating a Service

- Extend the Service class
  - adapter class exists, IntentService handles a lot of the details
- override onStartCommand
  - return an int describing what system should do for starting service
  - START_NOT_STICKY, if system kills service don't restart
  - START_STICKY, if system kills service then recreate, but does not redeliver intent
  - START_REDELIVER_INTENT, if system kills service then recreate and redeliver last intent

SMSResponser

```
public class ResponserService extends Service {

    private static final String TAG = "SMS-Response-Service";

    //The Broadcast Intent
    // sent by the Android system
    // when a SMS is received.
    private static final String RECEIVED_ACTION
            = "android.provider.Telephony.SMS_RECEIVED";

    private static final String SENT_ACTION = "SMS_SENT";
    private static final String DELIVERED_ACTION = "DELIVERED_SMS";

    private String requester;
    private String reply;
    private SharedPreferences myprefs;
```

SMS Responder

```
//The Action fired by the Android-System when a SMS was received.
private static final String RECEIVED_ACTION
        = "android.provider.Telephony.SMS_RECEIVED";

private static final String SENT_ACTION = "SENT_SMS";
private static final String DELIVERED_ACTION = "DELIVERED_SMS";

private String requester;
private String reply;
private SharedPreferences myprefs;
```

SMS Responder - onCreate

```
@Override
public void onCreate() {
    super.onCreate();
    myprefs = PreferenceManager.getDefaultSharedPreferences(this);

    registerReceiver(sentReceiver, new IntentFilter(SENT_ACTION));
    registerReceiver(deliverReceiver, new IntentFilter(DELIVERED_ACTION));

    IntentFilter receiverfilter = new IntentFilter(RECEIVED_ACTION);
    registerReceiver(receiver, receiverfilter);

    IntentFilter sendfilter = new IntentFilter(SENT_ACTION);
    registerReceiver(sender,sendfilter);
}
```

BROADCAST RECEIVERS

Broadcast Receivers

- The third of four application components
  - activities, services, broadcast receivers, content providers / receivers
- "A *broadcast receiver* is a component that responds to system-wide broadcast announcements."
- Android system sends multiple kinds of broadcasts
  - screen turned off, battery low, picture captured, SMS received, SMS sent, and more

Broadcasts

- Another use of intents
- Intents used to start activities and services
  - startActivity()
  - startActivityForResult()
  - startService()
  - bindService()
- Also used to deliver information from system and applications to other applications
- via Broadcast Intents

BroadcastReceivers

- What broadcasts are available?
- Check the Intent class
- http://developer.android.com/reference/android/content/Intent.html
  - search for "Broadcast Action"
- Also look in android-sdk\platforms\<number>\data\broadcast_actions.txt

Broadcasts Listed in Intent class

String	ACTION_CAMERA_BUTTON	Broadcast Action: The "Camera Button" was pressed.
String	ACTION_CHOOSER	Activity Action: Display an activity chooser, allowing the user to pick what they want to before proceeding.
String	ACTION_CLOSE_SYSTEM_DIALOGS	Broadcast Action: This is broadcast when a user action should request a temporary system dialog to dismiss.
String	ACTION_CONFIGURATION_CHANGED	Broadcast Action: The current device `Configuration` (orientation, locale, etc) has changed.
String	ACTION_CREATE_SHORTCUT	Activity Action: Creates a shortcut.
String	ACTION_DATE_CHANGED	Broadcast Action: The date has changed.
String	ACTION_DEFAULT	A synonym for `ACTION_VIEW`, the "standard" action that is performed on a piece of data.
String	ACTION_DELETE	Activity Action: Delete the given data from its container.
String	ACTION_DEVICE_STORAGE_LOW	Broadcast Action: A sticky broadcast that indicates low memory condition on the device This is a protected intent that can only be sent by the system.

Clicker

- Can you app send out any Broadcasts it wants?

A. yes

B. no

Broadcasts

- from broadcast_ actions.txt in sdk files
- platforms-> <api level>-> data\

```
android.intent.action.TIME_SET
android.intent.action.TIME_TICK
android.intent.action.UID_REMOVED
android.intent.action.USER_PRESENT
android.intent.action.WALLPAPER_CHANGED
android.media.ACTION_SCO_AUDIO_STATE_UPDATED
android.media.AUDIO_BECOMING_NOISY
android.media.RINGER_MODE_CHANGED
android.media.SCO_AUDIO_STATE_CHANGED
android.media.VIBRATE_SETTING_CHANGED
android.media.action.CLOSE_AUDIO_EFFECT_CONTROL_SESSION
android.media.action.OPEN_AUDIO_EFFECT_CONTROL_SESSION
android.net.conn.BACKGROUND_DATA_SETTING_CHANGED
android.net.wifi.NETWORK_IDS_CHANGED
android.net.wifi.RSSI_CHANGED
android.net.wifi.SCAN_RESULTS
android.net.wifi.STATE_CHANGE
android.net.wifi.WIFI_STATE_CHANGED
android.net.wifi.p2p.CONNECTION_STATE_CHANGE
android.net.wifi.p2p.PEERS_CHANGED
android.net.wifi.p2p.STATE_CHANGED
android.net.wifi.p2p.THIS_DEVICE_CHANGED
android.net.wifi.supplicant.CONNECTION_CHANGE
android.net.wifi.supplicant.STATE_CHANGE
android.provider.Telephony.SIM_FULL
android.provider.Telephony.SMS_CB_RECEIVED
android.provider.Telephony.SMS_EMERGENCY_CB_RECEIVED
android.provider.Telephony.SMS_RECEIVED
android.provider.Telephony.SMS_REJECTED
android.provider.Telephony.WAP_PUSH_RECEIVED
android.speech.tts.TTS_QUEUE_PROCESSING_COMPLETED
android.speech.tts.engine.TTS_DATA_INSTALLED
```

Broadcast Intents

- You can create your own Broadcasts Intents
- Many Intents listed in the Intent class are *protected intents* that may only be send by the system

public static final String **ACTION_TIMEZONE_CHANGED**

Added in

Broadcast Action: The timezone has changed. The intent will have the following extra values:

- *time-zone* - The java.util.TimeZone.getID() value identifying the new time zone.

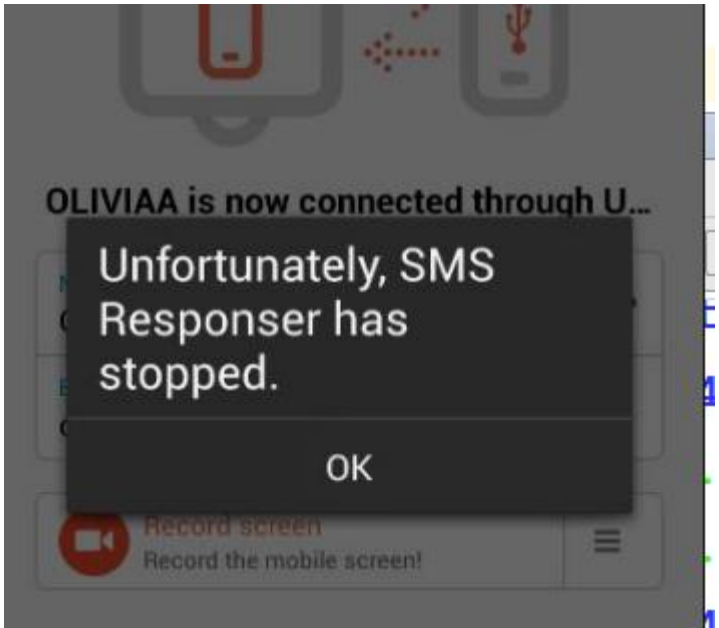
> This is a protected intent that can only be sent by the system.

Constant Value: "android.intent.action.TIMEZONE_CHANGED"

Protected Broadcasts

- Try it anyway??

```
private void attemptToSendTimeZoneChange() {
    Intent timeZoneChange =
            new Intent(Intent.ACTION_TIMEZONE_CHANGED);
    sendBroadcast(timeZoneChange);
}
```



03-23 13:16:50.222 388-397/? W/ActivityManager: Permission Denial: not allowed to send broadcast android.intent.action.TIMEZONE_CHANGED from pid=3470, uid=10140

Permissions Again

- Recall …
- Android 6.0, Marshmallow, API level 23 introduced changes to permissions
- Dangerous vs. Normal permissions
- Necessary to request Dangerous Permissions at runtime, not install time
- Listening for SMS send and receive Broadcasts is a Dangerous Permission

Receiving Broadcasts

- Activities and Services can listen for Broadcasts
- 4 steps

1. subclass BroadcastRecevier (implement onReceive method)
2. create IntentFilter object to specify the kinds of Broadcasts you want
3. register receiver (onResume() of Activity)
4. unregister receiver (onPause() of Activity)

-- alternatively use manifest to register receiver

Example: Step 1

- Start a service and print a toast at start up.
- Step one: subclass BroadcastReceiver

```
public class OnBootReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Log.d("OnBootReceiver", "Boot broadcast received!");
        Intent in = new Intent(context, DummyService.class);
        context.startService(in);
    }
}
```

Example: Step 2

- Step 2: Create intent filter to listen for broadcast
- In manifest

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


<receiver
    android:name=".OnBootReceiver"
    android:enabled="true"
    android:exported="true"
    android:permission="android.permission.RECEIVE_BOOT_COMPLETED" >
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED"
    </intent-filter>
</receiver>
```

Step 2 - More on Intent Filters

- For some Broadcasts you **cannot** use manifest to create IntentFilter
- must be done programmatically

public static final String **ACTION_TIME_TICK**

Added in AP

Broadcast Action: The current time has changed. Sent every minute. You *cannot* receive this through components declared in manifests, only by explicitly registering for it with `Context.registerReceiver()`.

> This is a protected intent that can only be sent by the system.

Constant Value: "android.intent.action.TIME_TICK"

Example: Step 3

- register receiver
- normally done in onResume of activity
- but no Activity on start up
- so, BroadcastReceiver in manifest file
- registers with System when app installed
- app must be started once for this to work

Spoofing Startup

- Painful to shut down and start up device
- Possible to spoof broadcasts
- recommend using emulator
- go to terminal
- adb shell

```
C:\Users\scottm\AndroidStudioProjects\Service_Example_SMS_Responder>
root@android:/ # am broadcast -a android.intent.action.BOOT_COMPLETE
am broadcast -a android.intent.action.BOOT_COMPLETED
Broadcasting: Intent { act=android.intent.action.BOOT_COMPLETED }
Broadcast completed: result=0
root@android:/ #
```

BROADCAST RECEIVERS IN AUTO TEXTING APP

In SMS Responder

- Service has inner classes for BroadcastReceiver that listens for Broadcast of SMS message received
- create and register receivers when service stated
- unregister when service destroyed
- **key point: override the onReceive method for BroadcastReceiver subclass**

SMS Received Broadcast

- from
- developer.android.com/reference/android/provider/Telephony.Sms.Intents.html

```
//The Broadcast Intent
// sent by the Android system
// when a SMS is received.
private static final String RECEIVED_ACTION
        = Telephony.Sms.Intents.SMS_RECEIVED_ACTION;
```

SMS Responder Service

- Create and register receivers

```
public void onCreate() {
    super.onCreate();
    Log.d(TAG, "In onCreate for ResponserService class");
    myprefs = PreferenceManager.getDefaultSharedPreferences(this);

    // sentReceiver informed when message from our app sent
    registerReceiver(sentReceiver, new IntentFilter(SENT_ACTION));

    // deliverReceiver informed when message from our app delivered
    registerReceiver(deliverReceiver, new IntentFilter(DELIVERED_ACTION));

    // receiver is the Broadcast receiver that actually responds to
    // incoming SMS messages
    IntentFilter receiverfilter = new IntentFilter(RECEIVED_ACTION);
    registerReceiver(receiver, receiverfilter);
```

SMS Received - Broadcast Receiver

```
BroadcastReceiver receiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context c, Intent in) {
        Log.v(TAG,"On Receive");
        if(in.getAction().equals(RECEIVED_ACTION)) {
            Log.v(TAG,"On SMS RECEIVE");
            Bundle bundle = in.getExtras();
            if(bundle!=null) {
                Object[] pdus = (Object[])bundle.get("pdus");
                SmsMessage[] messages = new SmsMessage[pdus.length];
                for(int i = 0; i<pdus.length; i++) {
                    Log.v(TAG,"FOUND MESSAGE");
                    messages[i]=SmsMessage.createFromPdu((byte[])pdus[i]);
                }
                for(SmsMessage message: messages)
                    requestReceived(message.getOriginatingAddress());
                respond();
            }
```

SMS Data

- The SMS data in the Bundle (map) is under the key "pdus"
  - pdu, protocol data unit (some sources indicate protocol description unit)

respond method

```
private void respond() {
    reply = myprefs.getString("reply", "Thank you for your message. I
    if(reply.length() == 0)
        reply = "Thank you for your message. I am busy now. I will ca
    SmsManager sms = SmsManager.getDefault();
    Intent sentIn = new Intent(SENT_ACTION);
    PendingIntent sentPIn
        = PendingIntent.getBroadcast(this, 0, sentIn, 0);

    Intent deliverIn = new Intent(DELIVERED_ACTION);
    PendingIntent deliverPIn
        = PendingIntent.getBroadcast(this, 0, deliverIn, 0);

    if(reply.length() > 140)
        reply = reply.substring(0, 140);

    sms.sendTextMessage(requester, null, reply, sentPIn, deliverPIn);
```

PendingIntent

- Intent to deliver when some criteria met

```
dingIntent sentPIn
 = PendingIntent.getBroadcast(this, 0, sentIn, 0);
```

- Parameters:
- this = context in which PendingIntent should sendBroadcast
- 0 = private request code for sender
- sentIn = Intent to be Broadcast
- 0 = flags to modify send behavior

SMSManager.sendTextMessage

```
sms.sendTextMessage(requester, null, reply, sentPIn, deliverPIn);
```

- address to send text to (destination, recipient)
- service center address (null = default)
- text of message
- pending intent to deliver when message sent
- pending intent to deliver when message delivered

BroadcastReceiver for Sent

```
private BroadcastReceiver sentReceiver = new BroadcastReceiver() {

@Override public void onReceive(Context c, Intent in) {
    Log.d(TAG, "in onReceive method of sentReceiver");
    Log.d(TAG, "result code: " + getResultCode());
    Log.d(TAG, "result code equals Activity.RESULT_OK " -
    Log.d(TAG, "Context: " + c);
    Log.d(TAG, "Intent: " + in);
    if(getResultCode() == Activity.RESULT_OK
            && in.getAction().equals(SENT_ACTION)) {
        // SMS Sent was from our app
        Log.d(TAG, "Activity result ok");
        smsSent();
    }
    else {
        Log.d(TAG, "Either result not okay, or SMS sent w
        smsFailed();
    }
```

SMS Sent

- Notify User with a Toast
  - Probably better to use Notification
  - Toast used so we see app in action during demonstration

```
public void smsSent(){
    Toast.makeText(this,
            "Auto Responding to message. SMS sent'
            Toast.LENGTH_SHORT).show();
}
```

Unregistering Receivers

- When no longer need unregister your receivers
- In this case when the service is shut down

```
@Override
public void onDestroy() {
    super.onDestroy();
    Log.d(TAG, "in onDestroy");
    unregisterReceiver(receiver);
    // unregisterReceiver(sender);
    unregisterReceiver(sentReceiver);
    unregisterReceiver(deliverReceiver);
}
```

INITIATING BROADCASTS OURSELVES

Broadcast Receivers

- Applications can initiate broadcasts to inform other applications of status or readiness
- Don't display UI
  - may create status bar notifications
- Usually just a gateway to other components and does very minimal work
  - initiate service based on some event
- Broadcasts are delivered as Intents

Broadcast Receivers

- intents sent by sendBroadcast() method
- LocalBroadcastManager to send Broadcasts within your application only

More on Broadcast Receivers

- can't initiate asynchronous actions in onReceive
  - like creating and starting an AsyncTask
  - because when method done BroadcastReceiver no longer active and system can and will kill the process
- ***May need to use Notification Manager or start a service***

Stopping Service

- Once started service runs until device shut down
- Add option to start and shut down the service
- Could add capability to start service on device start up