CS378 - Mobile Computing

Services and Broadcast Receivers
Services

• One of the four primary application components:
  – activities
  – content providers
  – services
  – broadcast receivers
Services

• 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
Forms of Services

• Stated:

  – application component, such as an Activity, starts the service with the method call 
    startService()

  – once started service can run in background indefinitely

  – generally services do not return a result (see bound service)

  – service should stop itself when done
Forms of Services

• 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

• private service (manifest) cannot be bound by other applications
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
Creating a Service

• create subclass of Android Service class or one of its existing subclasses
• override callback methods that handle important aspects of service lifecycle
• most important of these are:
  – `onStartCommand`
  – `startService`
  – `onBind`
  – `onCreate`
  – `onDestroy`
  – `stopSelf`
  – `stopService`
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 no called) service runs as long as at least one component bound to it
Service Lifecycle

- Call to startService():
  - onCreate()
  - onStartCommand()
- Service running:
  - The service is stopped by itself or a client
- onDestroy()
- Service shut down

- Call to bindService():
  - onCreate()
  - onBind()
- Clients are bound to service:
  - All clients unbind by calling unbindService()
  - onUnbind()
- onDestroy()
- Service shut down

Unbounded service

Bounded service
Service Example

• From Roger Wallace
  – 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?
Example Service Application

• From *The Android Developer's Cookbook*
• SMSResponder Application
• Response stored in shared preferences
• App simply allows changes to message
Using SMS

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

```xml
<uses-permission android:name="android.permission.RECEIVE_SMS" />
<uses-permission android:name="android.permission.SEND_SMS" />```
ResponseSMS Basic App

• All work done in onCreate method

```java
@Override
public void onCreate(Bundle savedInstanceState) {
    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);
    bt1 = (Button) this.findViewById(R.id.submit);
    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);
```
bt1.setOnClickListener(new OnClickListener()
{
    public void onClick(View view)
    {
        updater.putString("reply", ed1.getText().toString());
        updater.commit();
        SMSResponder.this.finish();
    }
});

try {
    // start Service
    Intent svc = new Intent(this, ResponserService.class);
    startService(svc);
}
catch (Exception e) {
    Log.e("onCreate", "service creation problem", e);
}
Service Running

app still running, and service has started
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

See all your apps.
Touch the Launcher icon.

This is a test text message from one emulator to another.
Emulator Texts

(from one emulator to another.)

Google

15555215554: This is a test text message from one emulator to another.

Sent: 11:42PM

Type to compose
Testing Service

1-555-521-5554: Thank you for your message. I am busy now. I will call you later
Sent: 12:04AM

Me: Hey!
Sent: 12:06AM

1-555-521-5554: Sorry I am busy. I will get back to you soon.
Sent: 12:06AM

Me: Test
Sent: 12:07AM

1-555-521-5554: Sorry I am

1-555-521-5556: Test again
Sent: Apr 22

1-555-521-5556: Test again.
Sent: 12:02AM

1-555-521-5556: Test
Sent: 12:03AM

1-555-521-5556: Test Test tests test
Sent: 12:04AM

1-555-521-5556: Hey!
Sent: 12:06AM
Creating a Service

• Extend the Service class
  – adapter class exists, IntentService that 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
// 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;
@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

• The fourth main application component
• "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
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 to perform based on some event
• Broadcasts are delivered as Intents
Broadcast Receivers

• receive intents sent by sendBroadcast() method

• LocalBroadcastManager to send Broadcasts within your application only

• In SMS responder register receivers

• unregister when service destroyed

• **key point: override the onReceive method for BroadcastReceiver subclass**
BroadcastReceivers

• What broadcasts are available?
• Check the Intent class
• Also look in android-sdk\platforms\<number>\data\broadcast_actions.txt
### Broadcasts

<table>
<thead>
<tr>
<th>String</th>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTION_CAMERA_BUTTON</td>
<td>Broadcast Action: The &quot;Camera Button&quot; was pressed.</td>
<td></td>
</tr>
<tr>
<td>ACTION_CHOOSER</td>
<td>Activity Action: Display an activity chooser, allowing the user to pick what they want to before proceeding.</td>
<td></td>
</tr>
<tr>
<td>ACTION_CLOSE_SYSTEM_DIALOGS</td>
<td>Broadcast Action: This is broadcast when a user action should request a temporary system dialog to dismiss.</td>
<td></td>
</tr>
<tr>
<td>ACTION_CONFIGURATION_CHANGED</td>
<td>Broadcast Action: The current device Configuration (orientation, locale, etc) has changed.</td>
<td></td>
</tr>
<tr>
<td>ACTION_CREATE_SHORTCUT</td>
<td>Activity Action: Creates a shortcut.</td>
<td></td>
</tr>
<tr>
<td>ACTION_DATE_CHANGED</td>
<td>Broadcast Action: The date has changed.</td>
<td></td>
</tr>
<tr>
<td>ACTION_DEFAULT</td>
<td>A synonym for ACTION_VIEW, the &quot;standard&quot; action that is performed on a piece of data.</td>
<td></td>
</tr>
<tr>
<td>ACTION_DELETE</td>
<td>Activity Action: Delete the given data from its container.</td>
<td></td>
</tr>
<tr>
<td>ACTION_DEVICE_STORAGE_LOW</td>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>
**Broadcasts**

- from broadcast_actions.txt in sdk files
- platforms-><api level>->data\n
<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>android.intent.action.TIME_SET</td>
<td>Time set event triggered by the system clock.</td>
</tr>
<tr>
<td>android.intent.action.TIME_TICK</td>
<td>Time tick event triggered by the system clock.</td>
</tr>
<tr>
<td>android.intent.action.UID_REMOVED</td>
<td>UID removed event triggered by the system clock.</td>
</tr>
<tr>
<td>android.intent.action.USER_PRESENT</td>
<td>User present event triggered by the system clock.</td>
</tr>
<tr>
<td>android.intent.action.WALLPAPER_CHANGED</td>
<td>Wallpaper changed event triggered by the system clock.</td>
</tr>
<tr>
<td>android.media.ACTION_SCO_AUDIO_STATE_UPDATED</td>
<td>SCO audio state updated event.</td>
</tr>
<tr>
<td>android.media.AUDIO_BECOMING_NOISY</td>
<td>Audio becoming noisy event triggered by the system clock.</td>
</tr>
<tr>
<td>android.media.RINGER_MODE_CHANGED</td>
<td>Ringer mode changed event triggered by the system clock.</td>
</tr>
<tr>
<td>android.media.SCO_AUDIO_STATE_CHANGED</td>
<td>SCO audio state changed event triggered by the system clock.</td>
</tr>
<tr>
<td>android.media.VIBRATE_SETTING_CHANGED</td>
<td>Vibrate setting changed event triggered by the system clock.</td>
</tr>
<tr>
<td>android.media.action.CLOSE_AUDIO_EFFECT_CONTROL_SESSION</td>
<td>Close audio effect control session event triggered by the system clock.</td>
</tr>
<tr>
<td>android.media.action.OPEN_AUDIO_EFFECT_CONTROL_SESSION</td>
<td>Open audio effect control session event triggered by the system clock.</td>
</tr>
<tr>
<td>android.net.conn.BACKGROUND_DATA_SETTING_CHANGED</td>
<td>Background data setting changed event triggered by the system clock.</td>
</tr>
<tr>
<td>android.net.wifi.NETWORK_IDS_CHANGED</td>
<td>Wi-Fi network IDs changed event triggered by the system clock.</td>
</tr>
<tr>
<td>android.net.wifi.RSSI_CHANGED</td>
<td>Wi-Fi signal strength changed event triggered by the system clock.</td>
</tr>
<tr>
<td>android.net.wifi_SCAN_RESULTS</td>
<td>Wi-Fi scan results event triggered by the system clock.</td>
</tr>
<tr>
<td>android.net.wifi.STATE_CHANGE</td>
<td>Wi-Fi state changed event triggered by the system clock.</td>
</tr>
<tr>
<td>android.net.wifi.WIFI_STATE_CHANGED</td>
<td>Wi-Fi state changed event triggered by the system clock.</td>
</tr>
<tr>
<td>android.net.wifi.p2p.CONNECTION_STATE_CHANGE</td>
<td>Wi-Fi p2p connection state changed event triggered by the system clock.</td>
</tr>
<tr>
<td>android.net.wifi.p2p.PEERS_CHANGED</td>
<td>Wi-Fi p2p peers changed event triggered by the system clock.</td>
</tr>
<tr>
<td>android.net.wifi.p2p.STATE_CHANGED</td>
<td>Wi-Fi p2p state changed event triggered by the system clock.</td>
</tr>
<tr>
<td>android.net.wifi.p2p.THIS_DEVICE_CHANGED</td>
<td>Wi-Fi this device changed event triggered by the system clock.</td>
</tr>
<tr>
<td>android.net.wifi.supplicant.CONNECTION_CHANGE</td>
<td>Wi-Fi supplicant connection change event triggered by the system clock.</td>
</tr>
<tr>
<td>android.net.wifi.supplicant.STATE_CHANGE</td>
<td>Wi-Fi supplicant state change event triggered by the system clock.</td>
</tr>
<tr>
<td>android.provider.Telephony.SIM_FULL</td>
<td>SIM full event triggered by the system clock.</td>
</tr>
<tr>
<td>android.provider.Telephony.SMS_CB_RECEIVED</td>
<td>SMS callback received event triggered by the system clock.</td>
</tr>
<tr>
<td>android.provider.Telephony.SMS_EMERGENCY_CB_RECEIVED</td>
<td>SMS emergency callback received event triggered by the system clock.</td>
</tr>
<tr>
<td>android.provider.Telephony.SMS_RECEIVED</td>
<td>SMS received event triggered by the system clock.</td>
</tr>
<tr>
<td>android.provider.Telephony.SMS_REJECTED</td>
<td>SMS rejected event triggered by the system clock.</td>
</tr>
<tr>
<td>android.provider.Telephony.WAP_PUSH_RECEIVED</td>
<td>WAP push received event triggered by the system clock.</td>
</tr>
<tr>
<td>android.speech.tts.TTS_QUEUE_PROCESSING_COMPLETED</td>
<td>TTS queue processing completed event triggered by the system clock.</td>
</tr>
<tr>
<td>android.speech.tts.engine.TTS_DATA_INSTALLED</td>
<td>TTS data installed event triggered by the system clock.</td>
</tr>
</tbody>
</table>
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

• incoming SMS messages trigger respond method

```java
private void respond() {
    reply = myprefs.getString("reply", "Thank you for your message. I am busy
    if(reply.length() == 0)
        reply = "Thank you for your message. I am busy now. I will call you !"
    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);
```

Stopping Service

• Once started service runs until device shut down
• Starts again when app started again
• Add option to start and shut down the service
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);
        }
    }
}
private boolean isMyServiceRunning() {
    Log.v(TAG, "checking if service is running");
    ActivityManager manager
        = (ActivityManager) getSystemService(ACTIVITY_SERVICE);
    for (RunningServiceInfo service :
            manager.getRunningServices(Integer.MAX_VALUE)) {
        Log.v(TAG, service.service.getClassName() + ":");
        if (serviceName.equals(service.service.getClassName())) {
            return true;
        }
    }
    return false;
}