Android Application Development, COMP 10073

Mohawk College, Winter 2021

Issues with Broadcasts

https://developer.android.com/guide/components/broadcasts.html

As we saw with the previous SMS example, and application can be designed to respond even though it is not running. Be careful not to abuse the opportunity to respond to broadcasts and run jobs in the background that can contribute to a slow system performance.

Problems occur with:

- Low memory devices
- Lots of background processes

When a large number of processes wake up at the same time:

- Must be swapped in and out of ram.
- Results in memory thrashing.
- Poor performance for the app the user it trying to use.

A large number of services can trigger at once in response to an implicit broadcast. A common instance of this is with activity_change. On some devices, 40 apps can wakeup from this broadcast. When this broadcast is sent frequently, the devices CPU cycles are consumed, producing a sluggish experience.

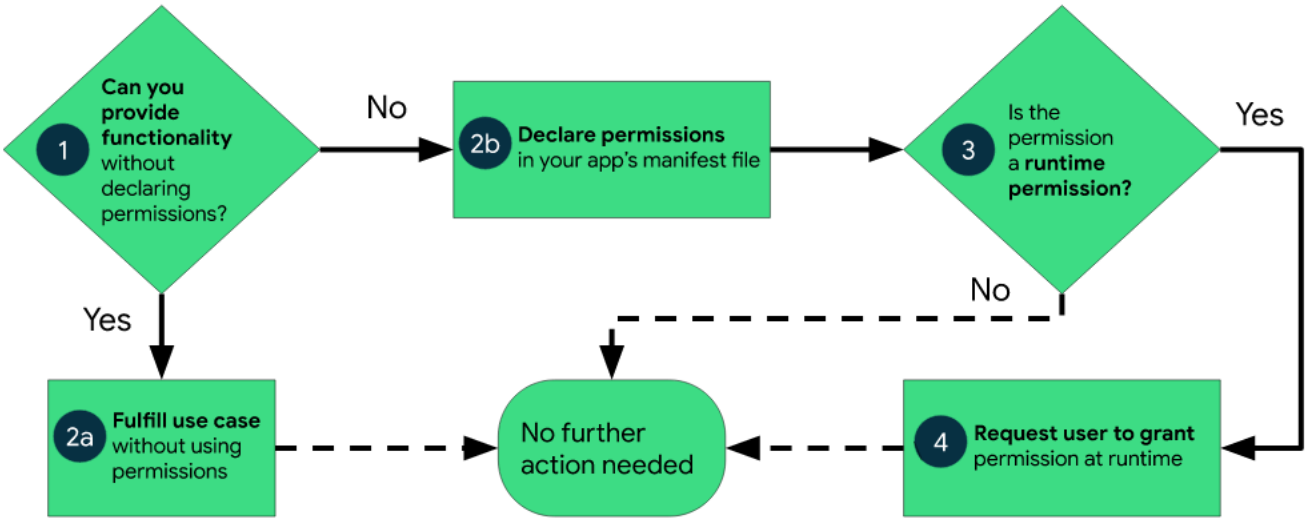
In Android 8.0 you cannot use the manifest to declare a receiver for most implicit broadcasts (broadcasts that don't target your app specifically). This means that our SMS message example won't work in the same way on API level 26 and above, the app will only respond to SMS messages when the user is actively using your app.

In Android 7.0 (API level 24) picture and video broadcasts were removed to improve general performance. Allowing applications to register as receivers for these broadcasts will slow down core features of the phone, picture taking. You've likely had the experience of using a photo app that has a slow response time, and within the 1/2 second or so delay your subject has moved (esp. a cat).

Types of Permissions

https://developer.android.com/guide/topics/permissions/overview

In the previous discussion we mentioned that being able to intercept an SMS message is considered to be a "Dangerous Permission" by the Android system.



Normal permissions: allow access to data and actions that extend beyond your app's sandbox. However, the data and actions present very little risk to the user's privacy, and the operation of other apps.

Runtime permissions, also known as **dangerous permissions**, give your app additional access to restricted data, and they allow your app to perform restricted actions that more substantially affect the system and other apps. Therefore, you need to request runtime permissions in your app before you can access the restricted data or perform restricted actions. Special permissions

Special permissions correspond to particular app operations. Only the platform and OEMs can define special permissions. Additionally, the platform and OEMs usually define special permissions when they want to protect access to particularly powerful actions, such as drawing over other apps.

Best practices: App permissions support the following user privacy goals:

Control: The user has control over the data that they share with apps.

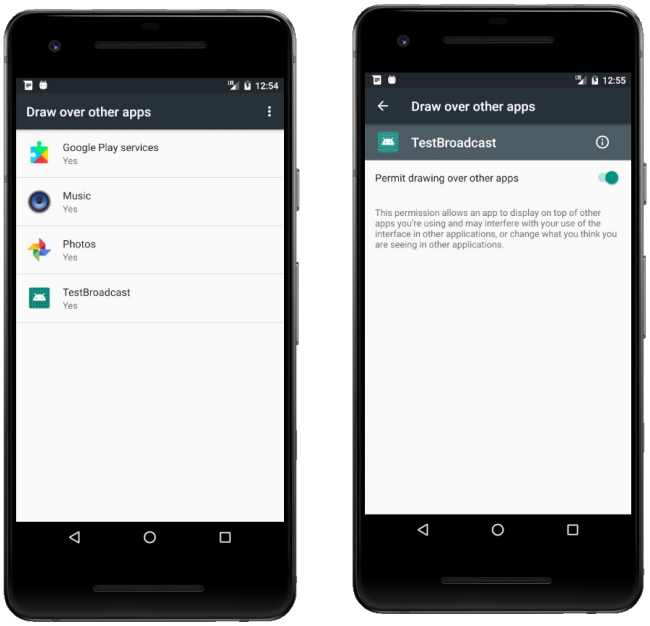
Transparency: The user understands why the app accesses certain data.

Data minimization: An app accesses only the data that's required.

Screen Overlay Permission

Previously our app ran into an issue where it would sometimes complain about some issue with overlays. Adding the SYSTEM_ALERT_WINDOW permission to the manifest file includes our demo on the “Draw over other apps” list, like so:

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



While you can give yourself this permission the problem is really that the messenger app is placing a notification on top of our app, so... this doesn’t really fix things for us. Another way to handle this is to ask for the permission once when your application starts, which won’t always coincide with the receipt of a text message.

Requesting Permissions

https://developer.android.com/training/permissions/requesting

Before you ask for a permission you can check to see if you already have it:

```
if (ActivityCompat.checkSelfPermission(this,
        Manifest.permission.RECEIVE_SMS) !=
        PackageManager.PERMISSION_GRANTED) {
    ActivityCompat.requestPermissions(this,
            new String[]{Manifest.permission.RECEIVE_SMS}, 1);
}
```

The request is asynchronous, so you can’t confirm immediately whether or not it was granted. Try for example adding this code after the requestPermissions call.

```
    if (ActivityCompat.checkSelfPermission(this,
            Manifest.permission.RECEIVE_SMS) ==
            PackageManager.PERMISSION_GRANTED) {
        Log.d(TAG, "SMS perms Granted");
    } else {
        Log.d(TAG, "SMS perms *NOT* Granted");
    }
```

Location Manager

https://developer.android.com/reference/android/location/LocationManager

This class provides access to the system location services. These services allow applications to obtain periodic updates of the device's geographical location, or to be notified when the device enters the proximity of a given geographical location.

Unless otherwise noted, all Location API methods require the Manifest.permission.ACCESS_COARSE_LOCATION or Manifest.permission.ACCESS_FINE_LOCATION permissions. If your application only has the coarse permission then providers will still return location results, but the exact location will be obfuscated to a coarse level of accuracy.

Add the following permission

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

Modify the onCreate method so it checks for the location permission, and either adds it if it isn't available, or enables the location manager if it is.

```
public class MainActivity extends AppCompatActivity
    implements LocationListener {
    public static String TAG = "==MainActivity==";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        if (ActivityCompat.checkSelfPermission(this,
                Manifest.permission.ACCESS_FINE_LOCATION) !=
                PackageManager.PERMISSION_GRANTED) {

            Log.d(TAG, "Requesting Permissions");
            ActivityCompat.requestPermissions(this,
                    new String[] {Manifest.permission.ACCESS_FINE_LOCATION},
                    1001 );
        } else {
            Log.d(TAG, "Have FINE_LOCATION Permission");
            LocationManager lm = (LocationManager)
                    getSystemService(Context.LOCATION_SERVICE);
            lm.requestLocationUpdates(LocationManager.GPS_PROVIDER,
                    5000, 0, this);
            lm.requestLocationUpdates(LocationManager.NETWORK_PROVIDER,
                    5000, 0, this);
        }
        Log.d(TAG, "onCreate");
    }
```

Implementing the Location Listener

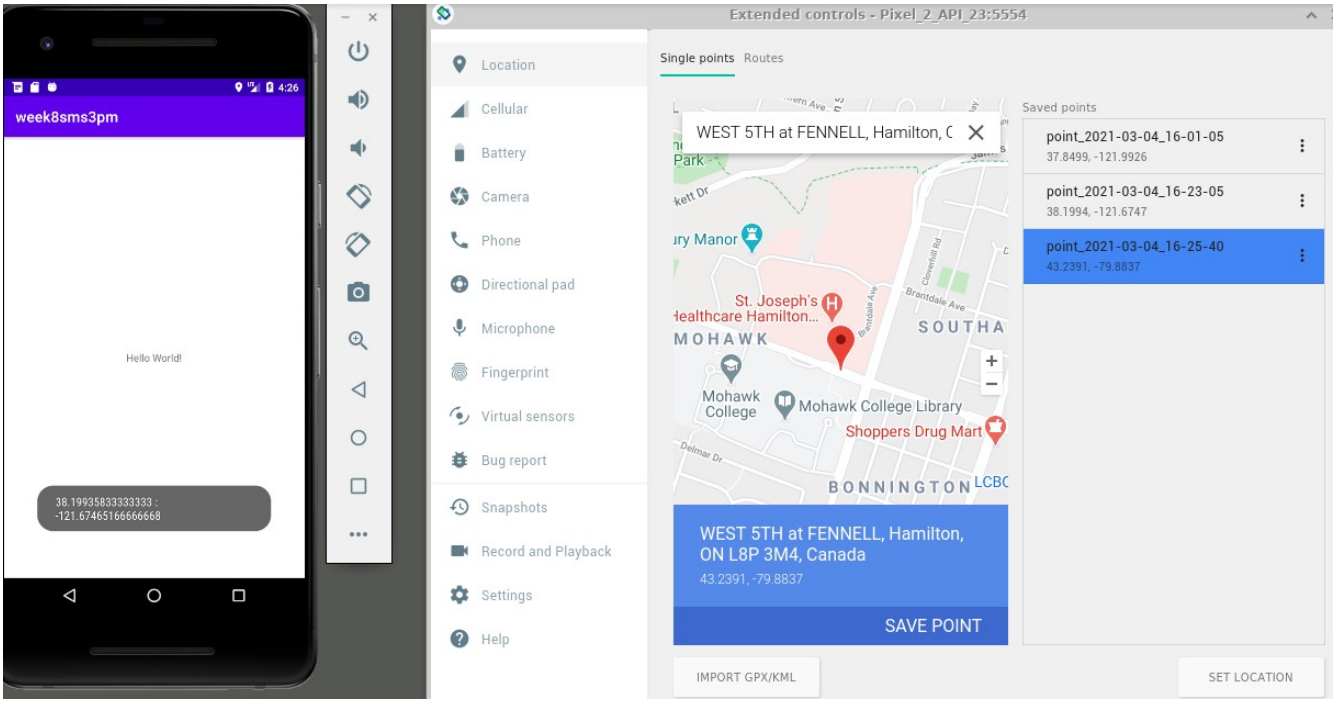
We need to override several methods to implement the location listener. The first 3 respond to changes in the provider status and can be empty. The last method will be called every 5 seconds and generate a Toast message showing the current location.

```
@Override
public void onStatusChanged(String provider, int status, Bundle extras) {}
@Override
public void onProviderEnabled(String provider) {}
@Override
public void onProviderDisabled(String provider) {}

@Override
public void onLocationChanged(Location location) {
    // Location Changed!
    double loclat = location.getLatitude();
    double loclong = location.getLongitude();

    Toast.makeText(this, loclat + " : " + loclong, Toast.LENGTH_LONG).show();
    Log.d(TAG, "onRequestPermissionsResult");
}
```

You can use the AVD to set the location of the device.



Can We Avoid Running the App Twice?

The current implementation needs to be run twice, which isn't great. The first time adds the permission, the second time the app is started it sets up the location manager. We could implement a button but it would be better if we could some how get confirmation that we had the permission we asked for.

The following method will be triggered after we accept permissions

```
@Override
public void onRequestPermissionsResult(int resultCode,
                                       String permissions[], int[] results ) {
    Log.d(TAG, "onRequestPermissionsResult");
    switch (resultCode) {
        case 1001:
            if (results.length > 0 && results[0] ==
                    PackageManager.PERMISSION_GRANTED) {

                LocationManager lm = (LocationManager)
                        getSystemService(Context.LOCATION_SERVICE);
                lm.requestLocationUpdates(LocationManager.GPS_PROVIDER,
                        5000, 0, this);
                lm.requestLocationUpdates(LocationManager.NETWORK_PROVIDER,
                        5000, 0, this);
            }
    }
}
```

Reusing Code Elements

https://developer.android.com/guide/topics/resources/string-resource

We mentioned briefly that constants in the strings.xml file can include parameters. Load up last weeks example and remove the hello fragment strings and replace them with this:

```
<string name="hello">Fragment %1$s The Switch is %2$s</string>
```

In the fragment code itself we can now use code like this to set the fragment text:

```
tv.setText(getString(R.string.hello, "ONE", "ON"));
```

Lab4 asked about fragment reusability. Many people talk generally about their application and don't focus on these prompts.

Consider what needs to be done to the previous lab to get the following layout.