Android Application Development

Android Application

AndroidManifest.xml

- Each Android project includes a manifest file, **AndroidManifest.xml**, stored in the root of its project hierarchy.
- The manifest **defines the structure** and **metadata** of your application, its components, and its requirements.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.myorg.twitterfeeds"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="10" />
    <uses-permission android:name="android.permission.INTERNET"/>
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".TwitterFeedsDemoActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

AndroidManifest.xml

- Includes nodes for each of the **Activities**, **Services**, **Content Providers**, and **Broadcast Receivers** that make up your application and, using **Intent Filters** and **Permissions**, determines **how they interact** with each other and with other applications.

- **Specifies application metadata** (such as its icon, version number, or theme), and additional top-level nodes **can specify any required permissions** and **define hardware, screen, or platform requirements**

- The manifest is made up of a **root manifest tag** with a package attribute set to the project's package. It should also include an **xmlns: android attribute** that supplies several system attributes used within the file.

AndroidManifest.xml

- **versionCode**: Define the **current application version as an integer** that increases with each version iteration
- **versionName**: Specify a **public version** that will be displayed to users
- **installLocation**: Specify whether to allow (or prefer) for your application be **installed on external storage** (usually an SD card) [preferExternal|auto]

AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.paad.myapp"
          android:versionCode="1"
          android:versionName="0.9 Beta"
          android:installLocation="preferExternal">
          [ ... manifest nodes ... ]
</manifest>
```

- **uses-sdk**: Enables **to define a minimum and maximum SDK version** that must be available on a device for your application to function properly, and **target SDK** for which it has been designed. [**minSDKVersion**, **maxSDKVersion**, and **targetSDKVersion**].
  - The minimum SDK version specifies the lowest version of the SDK. If not specified a minimum version, it defaults to 1, and your application crashes when it attempts to access unavailable APIs.
  - The target SDK version attribute enables you to specify the platform against which you did your development and testing. Good practice is to update the target SDK of your application to the latest platform release after you confirm it behaves as expected, even if you aren't making use of any new APIs.
  - maxSDKVersion: Unnecessary

AndroidManifest.xml

- **uses-configuration**: **Specifies each combination of input mechanisms** are supported by your application. Normally no need to include this node, though it can be useful for games that require particular input controls.
  - reqFiveWayNav — Specify true for this attribute if you require an input device capable of navigating up, down, left, and right and of clicking the current selection. This includes both trackballs and directional pads (D-pads).
  - reqHardKeyboard — If your application requires a hardware keyboard, specify true.
  - reqKeyboardType — Lets you specify the keyboard type as one of nokeys, qwerty, twelvekey, or undefined.
  - reqNavigation — Specify the attribute value as one of nonav, dpad, trackball, wheel, or undefined as a required navigation device.
  - reqTouchScreen — Select one of notouch, stylus, finger, or undefined to spec- ify the required touchscreen input.

```
<uses-configuration
    android:reqFiveWayNav="true"
    android:reqHardKeyboard="true"
    android:reqKeyboardType="qwerty"
    android:reqNavigation="trackball"
    android:reqTouchScreen="finger" />
```

AndroidManifest.xml

- **uses-feature** – Use multiple uses-feature nodes **to specify which hardware features your application requires**. This prevents your application from being installed on a device that does not include a required piece of hardware, such as NFC hardware, as follows:

  <uses-feature android:name=”android.hardware.nfc” />

You can require support for any hardware that is optional on a compatible device. Currently, optional hardware features include the following:

- Bluetooth
- Camera
- Location
- Microphone
- NFC
- Sensors
- Telephony
- Touchscreen
- USB
- Wi-Fi

```
<uses-feature android:name="android.hardware.camera" />
<uses-feature
    android:name="android.hardware.camera.autofocus"
    android:required="false" />
<uses-feature
    android:name="android.hardware.camera.flash"
    android:required="false" />
```

http://developer.android.com/guide/topics/manifest/uses-feature-element.html#features-reference.

AndroidManifest.xml

- **supports-screens** – Enables you to s**pecify the screen sizes your application has been designed and tested to**. On devices with supported screens, your application is laid out normally using the scaling properties associated with the layout files you supply.
  - smallScreens –typically, QVGA screens
  - normalScreens – HVGA, including WVGA and WQVGA.
  - largeScreens – Screens larger than normal.
  - xlargeScreens – Typically tablet devices.
- Honeycomb MR2 (API level 13) introduced additional attributes:
  - requiresSmallestWidthDp
  - compatibleWidthLimitDp
  - largestWidthLimitDp

```
<supports-screens
    android:largeScreens="true"
    android:normalScreens="true"
    android:smallScreens="false"
    android:xlargeScreens="true" />
```

AndroidManifest.xml

- uses-permission — (Security model), **declare the user permissions your application requires**. Each permission you specify will be presented to the user before the application is installed. Permissions are required for many APIs and method calls, generally those with an associated cost or security implication (such as dialing, receiving SMS, or using the location-based services).

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

- permission — Your application components can also create permissions to restrict access to shared application components.
  - Your application components can then create permissions by adding an android:permission attribute. Then you can include a uses-permission tag in your manifest to use these protected components, both in the application that includes the protected component and any other application that wants to use it.

```
<permission android:name="com.paad.DETONATE_DEVICE"
            android:protectionLevel="dangerous"
            android:label="Self Destruct"
            android:description="@string/detonate_description">
</permission>
```

AndroidManifest.xml

- application — A manifest can contain only one application node.
  - During development if you include a debuggable attribute set to true to enable debugging, then be sure to disable it for your release builds.
  - The application node also acts as a container for the Activity, Service, Content Provider, and Broadcast Receiver nodes that specify the application components. You specify the name of your custom application class using the android:name attribute.

```
<application android:icon="@drawable/icon"
             android:logo="@drawable/logo"
             android:theme="@android:style/Theme.Light"
             android:name=".MyApplicationClass"
             android:debuggable="true">
             [ ... application nodes ... ]
</application>
```

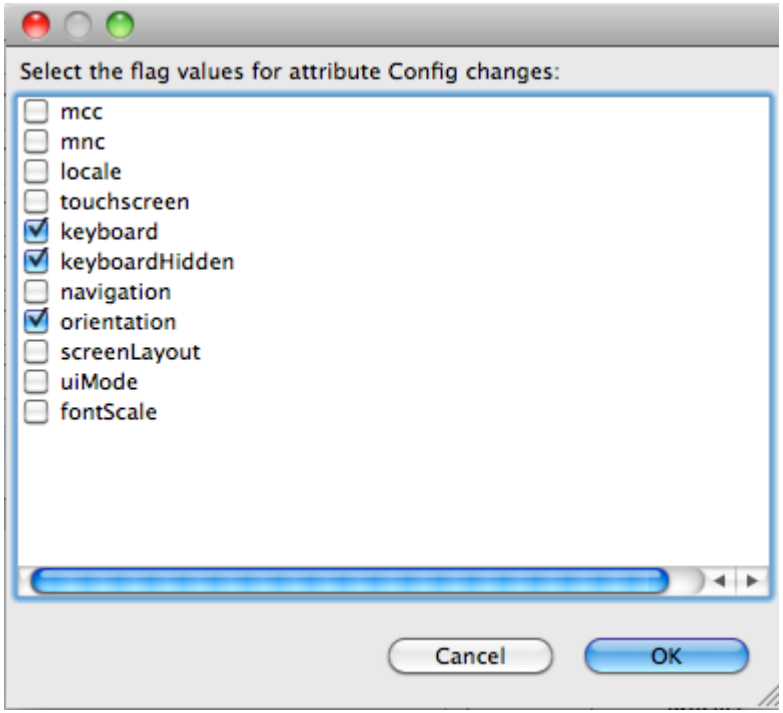
  - activity
  - service
  - provider
  - receiver
  - uses-library

Runtime Configuration Changes

- Android handles runtime changes to the language, location, and hardware by **terminating and restarting the active Activity**.
- In some special cases this default behavior may be inconvenient.
- To have an Activity listen for runtime configuration changes, add an android:configChanges attribute to its manifest node, specifying the configuration changes you want to handle.
  - mcc+mnc
  - locale
  - keyboardHidden
  - keyboard
  - fontScale
  - uiMode
  - orientation
  - screenLayout
  - screenSize
  - smallestScreenSize

Runtime Configuration Changes

- You can select multiple events you want to handle yourself



Test an activity with+without configChanges while changing orientation

```
<activity
    android:name=".MainScreenActivity"
    android:label="@string/app_name" android:configChanges="keyboard|keyboardHidden|orientation">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Runtime Configuration Changes

- Adding an android:configChanges attribute suppresses the restart for the specified configuration changes, instead triggering the **onConfigurationChanged** handler in the associated Activity. Override this method to handle the configuration changes yourself, using the passed-in **Configuration** object

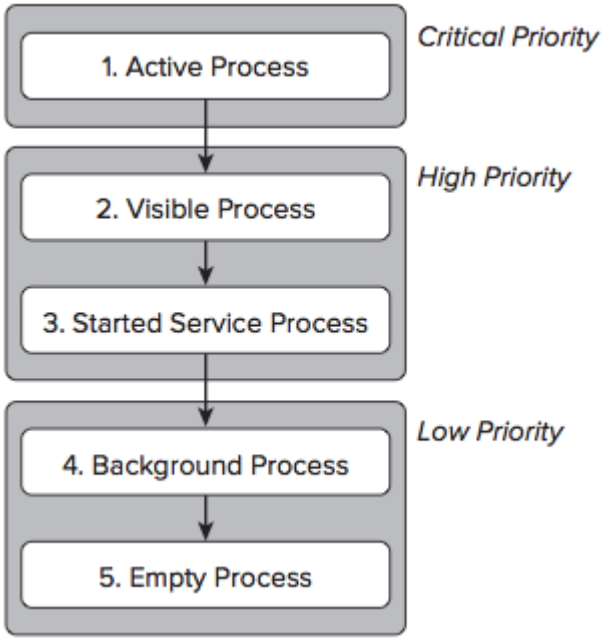
```
@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);
    // [ ... Update any UI based on resource values ... ]
    if (newConfig.orientation == Configuration.ORIENTATION_LANDSCAPE) {

    }
    if (newConfig.keyboardHidden == Configuration.KEYBOARDHIDDEN_NO) {

    }
}
```

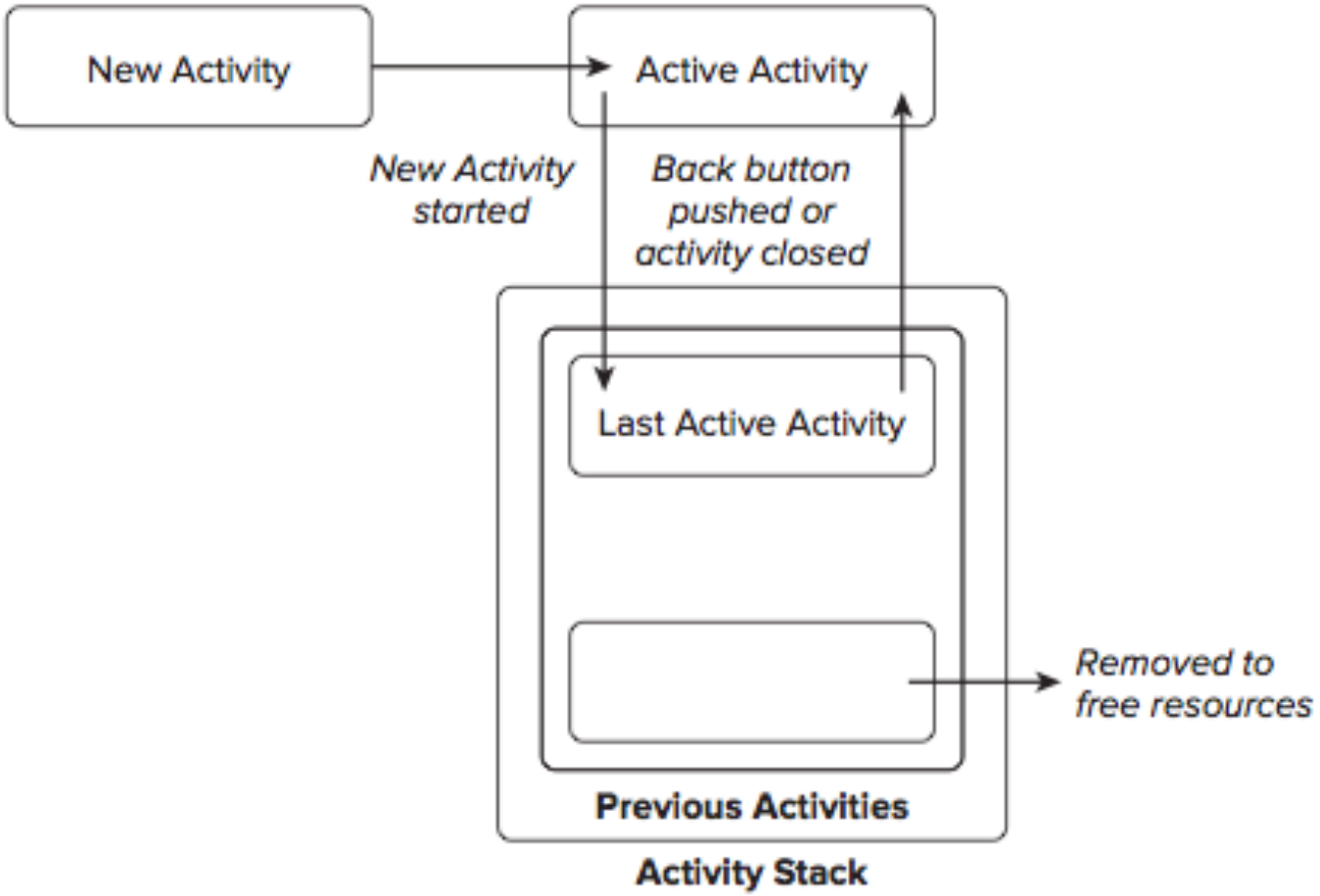
Android Application Lifecycle

- Application Class
- Events
- Application Lifecycle
- Application Singleton



Android Activities Revisited

Activity Stacks



Android Activities Revisited

Activity States