Device Administration

What Device Administration Is

- Enhanced Control Over Device
  - E.g., wipe
- Policy Enforcement
  - E.g., PIN/password criteria for system lock screen
- An Opt-In Capability by Users
  - Must install, then separately activate as a device administrator

What Device Administration Is Not

- Root
- Ways to Gain More Permissions
  - Dedicated API
  - Does not impact what permissions you hold or ability to perform permission-constrained operations
- Mobile Device Management (MDM)
  - MDM can **use** device administration, but device administration is not an MDM solution

Pieces of Device Administration

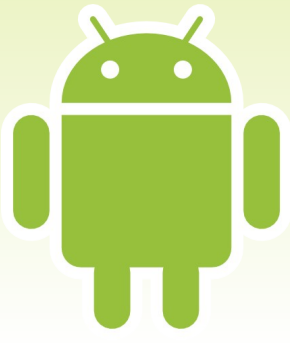
- DeviceAdminReceiver
  - Implementation, registered in manifest
  - Tied to metadata describing what you want to be able to do
  - Point for handling events related to device administration
    - Handling these events is not required

Pieces of Device Administration

- DevicePolicyManager
  - Used for proactively setting rules or performing actions
  - Settings are readable by any app
  - Must have a DeviceAdminReceiver and have app be configured by user as a device administrator to be able to modify things

DeviceAdminReceiver

- Base class that you extend
- Override methods for events you want to monitor
  - And are eligible for given policies
- Register in manifest
  - Actions tied to those methods/policies
  - `<meta-data>` element pointing to policies
  - **Require** `BIND_DEVICE_ADMIN` permission

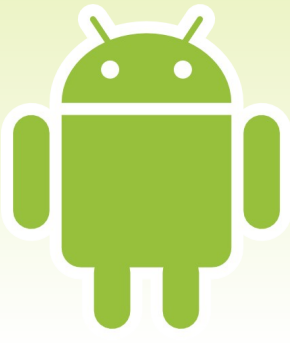


```
public class AdminReceiver extends DeviceAdminReceiver {
}
```

```
<receiver
  android:name="AdminReceiver"
  android:permission="android.permission.BIND_DEVICE_ADMIN">
  <meta-data
    android:name="android.app.device_admin"
    android:resource="@xml/device_admin"/>

  <intent-filter>
    <action android:name="android.app.action.DEVICE_ADMIN_ENABLED"/>
  </intent-filter>
</receiver>
```

Device Administration Metadata

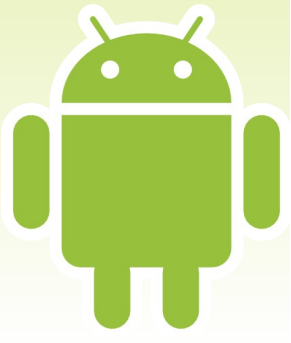
- XML resource
- `<device-admin>` root element, containing a `<uses-policies>` element
- Series of nested elements declaring which policies you wish to control
- User is presented with this list at the point of establishing your app as a device administrator
- You can only manipulate what you request here



```xml
<device-admin xmlns:android="http://schemas.android.com/apk/res/android">

  <uses-policies>
    <force-lock/>
  </uses-policies>

</device-admin>
```

Becoming a Device Administrator

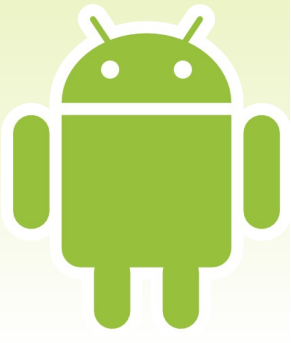
- Call `isAdminActive()` on DevicePolicyManager
  - If `false`, can create an `ACTION_ADD_DEVICE_ADMIN` Intent and use with `startActivity()` to route user to activate your device administrator status
    - `EXTRA_DEVICE_ADMIN` contains ComponentName of your DeviceAdminReceiver, to steer UI directly to your component
    - `EXTRA_ADD_EXPLANATION` for more info to show user



```
public class LockMeNowActivity extends Activity {
  private DevicePolicyManager mgr=null;
  private ComponentName cn=null;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.main);
    cn=new ComponentName(this, AdminReceiver.class);
    mgr=(DevicePolicyManager)getSystemService(DEVICE_POLICY_SERVICE);
  }

  public void lockMeNow(View v) {
    if (mgr.isAdminActive(cn)) {
      mgr.lockNow();
    }
    else {
      Intent intent=
          new Intent(DevicePolicyManager.ACTION_ADD_DEVICE_ADMIN);
      intent.putExtra(DevicePolicyManager.EXTRA_DEVICE_ADMIN, cn);
      intent.putExtra(DevicePolicyManager.EXTRA_ADD_EXPLANATION,
                      getString(R.string.device_admin_explanation));
      startActivity(intent);
    }
  }
}
```

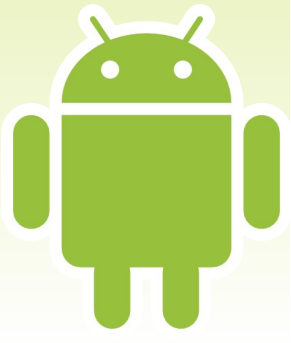
Metadata Policies

- `<encrypted-storage>`
  - Can call `setStorageEncryption()` to request that the device be encrypted
    - Still requires user acceptance, since this involves reboots and such
- `<disable-camera>`
  - Can call `setCameraDisabled()` to enable/disable access to camera



Metadata Policies

- `<disable-keyguard-features>`
  - Can call `setKeyguardDisabledFeatures()` to control what optional features are available on the lockscreen
    - Camera access
    - Lockscreen widgets
- `<force-lock>`
  - Can call `lockNow()` to force the device to lock, requiring entry of password on lockscreen
  - Call `setMaximumTimeToLock()`



Metadata Policies

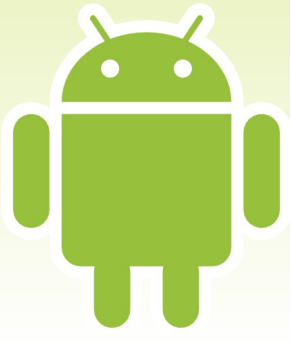
- `<expire-password>`
  - Can call `setPasswordExpirationTimeout()` to indicate when a new password must be set
- `<limit-password>`
  - Can call various methods to control the nature of passwords (e.g., minimum length)
- `<reset-password>`
  - Can call `resetPassword()` to reset password to known value

Metadata Policies

- `<watch-login>`
  - Can monitor for successful and failed password attempts
  - Triggers methods to be called on your DeviceAdminReceiver
- `<wipe-data>`
  - Can call `wipeData()` to wipe the device

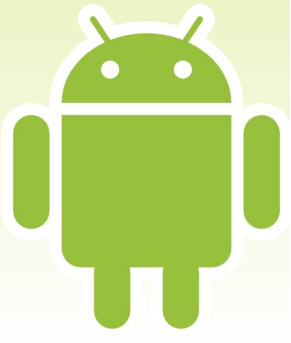
DeviceAdminReceiver Events

- `onDisableRequested()`
  - `ACTION_DEVICE_ADMIN_DISABLE_REQUESTED`
  - Called when user unchecks your component as being a device administrator
  - Return text of warning message (for dialog), or null otherwise
  - DO NOT OTHERWISE INTERFERE
    - Detect in-force policies and react to those
    - Do not try to block the disable operation



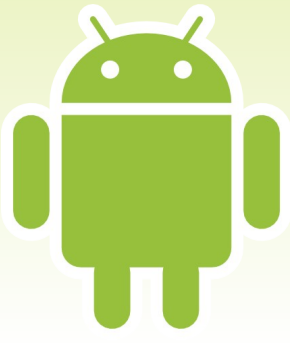
DeviceAdminReceiver Events

- `onDisabled()`
  - `ACTION_DEVICE_ADMIN_DISABLED`
  - Called when no longer a device admin
  - Does not mean your policies are not enforced
    - May be enforced by another device admin!
- `onEnabled()`
  - `ACTION_DEVICE_ADMIN_ENABLED`
  - Called when you become a device admin



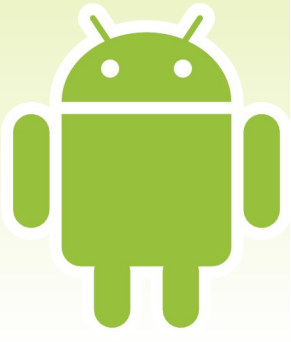
DeviceAdminReceiver Events

- `onPasswordChanged()`
  - `ACTION_PASSWORD_CHANGED`
  - Called when user changes their password
- `onPasswordExpiring()`
  - `ACTION_PASSWORD_EXPIRING`
  - Called periodically as getting near expiry time
    - On boot
    - Once/day
    - Once expired



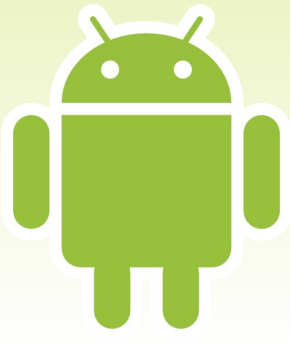
DeviceAdminReceiver Events

- `onPasswordFailed()`
  - `ACTION_PASSWORD_FAILED`
  - Requires `<watch-login>` policy
  - Called when the user enters the wrong password
- `onPasswordSucceeded()`
  - `ACTION_PASSWORD_SUCCEEDED`
  - Requires `<watch-login>` policy
  - Called when the use enters the correct password



Password Policies

- `setPasswordQuality()`
  - Controls if password required and how complex it must be (e.g., simple PIN?)
- Specific Password Criteria
  - `setPasswordMinimumLength()`
  - `setPasswordMinimumLetters()`
  - `setPasswordMinimumLowerCase()`
  - `setPasswordMinimumNonLetter()`
  - `setPasswordMinimumNumeric()`
  - `setPasswordMinimumSymbols()`
  - `setPasswordMinimumUpperCase()`

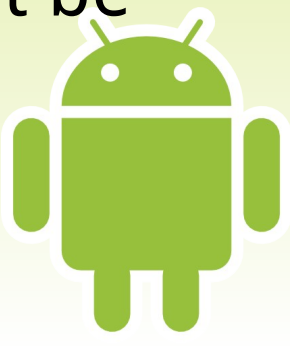


Password Policies

- `setPasswordExpirationTimeout()`
  - When does user have to provide a new password?
  - Helps enforce the other policies
    - Requiring a password with certain characteristics only takes effect when setting up next password
- `setPasswordHistoryLength()`
  - How many previous passwords should Android track, to prevent reuse?

Other DevicePolicyManager Stuff

- Getters
  - E.g., getPasswordQuality()
  - Used to report what the current requirements are
    - Based across all active device administrators
- `getStorageEncryptionStatus()`
  - Current actual status
  - Compared to `getStorageEncryption()`, which is requested encryption policy and may not yet be in force



Other DevicePolicyManager Stuff

- `isActivePasswordSufficient()`
  - Does the current password meet all criteria, including any you just established?

```
<receiver
  android:name="AdminReceiver"
  android:permission="android.permission.BIND_DEVICE_ADMIN">
  <meta-data
    android:name="android.app.device_admin"
    android:resource="@xml/device_admin"/>

  <intent-filter>
    <action android:name="android.app.action.DEVICE_ADMIN_ENABLED"/>
    <action android:name="android.app.action.ACTION_PASSWORD_CHANGED"/>
    <action android:name="android.app.action.ACTION_PASSWORD_FAILED"/>
    <action android:name="android.app.action.ACTION_PASSWORD_SUCCEEDED"/>
  </intent-filter>
</receiver>
```

```xml
<device-admin xmlns:android="http://schemas.android.com/apk/res/android">

  <uses-policies>
    <limit-password/>

    <watch-login/>
  </uses-policies>

</device-admin>
```

```
public class MainActivity extends Activity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    ComponentName cn=new ComponentName(this, AdminReceiver.class);
    DevicePolicyManager mgr=
        (DevicePolicyManager)getSystemService(DEVICE_POLICY_SERVICE);

    if (mgr.isAdminActive(cn)) {
      int msgId;

      if (mgr.isActivePasswordSufficient()) {
        msgId=R.string.compliant;
      }
      else {
        msgId=R.string.not_compliant;
      }

      Toast.makeText(this, msgId, Toast.LENGTH_LONG).show();
    }
    else {
      Intent intent=
          new Intent(DevicePolicyManager.ACTION_ADD_DEVICE_ADMIN);
      intent.putExtra(DevicePolicyManager.EXTRA_DEVICE_ADMIN, cn);
      intent.putExtra(DevicePolicyManager.EXTRA_ADD_EXPLANATION,
                      getString(R.string.device_admin_explanation));
      startActivity(intent);
    }

    finish();
  }
}
```

```
public class AdminReceiver extends DeviceAdminReceiver {
  @Override
  public void onEnabled(Context ctxt, Intent intent) {
    ComponentName cn=new ComponentName(ctxt, AdminReceiver.class);
    DevicePolicyManager mgr=
        (DevicePolicyManager)ctxt.getSystemService(Context.DEVICE_POLICY_SERVICE);

    mgr.setPasswordQuality(cn,
                           DevicePolicyManager.PASSWORD_QUALITY_ALPHANUMERIC);

    onPasswordChanged(ctxt, intent);
  }
```

```java
@Override
public void onPasswordChanged(Context ctxt, Intent intent) {
  DevicePolicyManager mgr=
      (DevicePolicyManager)ctxt.getSystemService(Context.DEVICE_POLICY_SERVICE);
  int msgId;

  if (mgr.isActivePasswordSufficient()) {
    msgId=R.string.compliant;
  }
  else {
    msgId=R.string.not_compliant;
  }

  Toast.makeText(ctxt, msgId, Toast.LENGTH_LONG).show();
}

@Override
public void onPasswordFailed(Context ctxt, Intent intent) {
  Toast.makeText(ctxt, R.string.password_failed, Toast.LENGTH_LONG)
       .show();
}

@Override
public void onPasswordSucceeded(Context ctxt, Intent intent) {
  Toast.makeText(ctxt, R.string.password_success, Toast.LENGTH_LONG)
       .show();
}
```