Permission model in Android >= 6

- If your app targets SDK 23 or higher, it prompts users to grant permissions at runtime, instead of install time
  - Your app must use the new permissions model!
- Users can revoke permissions at any time from the app Settings screen
- Your app needs to check that it has the permissions it needs every time it runs
- If your app runs on a device that has API 22 or lower, the app uses the old permissions model
  - When the user installs the app, they are prompted to grant all the permissions your app requests in its manifest
- **Note!** You can use the GitHub Google Samples EasyPermissions (or other more advanced/better libraries) which is a wrapper library to simplify basic system permissions logic when targeting SDK 23 or higher
  - [https://github.com/googlesamples/easypermissions](https://github.com/googlesamples/easypermissions)
Permission overview

• Declaring Permissions
  – The app declares all the permissions it needs in the manifest, as in earlier Android platforms

• Dangerous Permissions belong to Permission Groups
  – Permissions are divided into permission groups, based on their functionality

• Normal Permissions are Granted at Install Time
  – When the user installs or updates the app, the system grants the app all permissions listed in the manifest that fall under PROTECTION_NORMAL
  – For example, alarm clock and internet permissions fall under PROTECTION_NORMAL, so they are automatically granted at install time

• User Grants Permissions at Run-Time
  – When the app requests a permission, the system shows a dialog to the user, then calls the app's callback function to notify it whether the user granted the permission
Permission groups

- Related permissions are divided into permission groups to allow users to grant related permissions to an app in a single action.
- The user only has to grant permission once per app for each permission group. If the app subsequently requests a permission from the same permission group, the system automatically grants the permission without any action from the user.
- The system calls your app's onRequestPermissionsResult() method just as if the user had granted permission through the dialog box.
- See the permissions and permission groups table at: https://developer.android.com/training/permissions/index.html > System Permissions
Table 1. Dangerous permissions and permission groups.

<table>
<thead>
<tr>
<th>Permission Group</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALENDAR</td>
<td>READCALENDAR, WRITECALENDAR</td>
</tr>
<tr>
<td>CAMERA</td>
<td>CAMERA</td>
</tr>
<tr>
<td>CONTACTS</td>
<td>READCONTACTS, WRITECONTACTS, GETACCOUNTS</td>
</tr>
<tr>
<td>LOCATION</td>
<td>ACCESSFINELOCATION, ACCESSCOARSELOCATION</td>
</tr>
<tr>
<td>MICROPHONE</td>
<td>RECORD_AUDIO</td>
</tr>
<tr>
<td>PHONE</td>
<td>READ_PHONE_STATE, CALL_PHONE, READ_CALL_LOG, WRITE_CALL_LOG, ADD_VOICEMAIL, USE_SIP, PROCESS_OUTGOING_CALLS</td>
</tr>
<tr>
<td>SENSORS</td>
<td>BODY_SENSORS</td>
</tr>
<tr>
<td>SMS</td>
<td>SEND_SMS, RECEIVE_SMS, READ_SMS, RECEIVE_WAP_PUSH, RECEIVE_MMS</td>
</tr>
<tr>
<td>STORAGE</td>
<td>READEXTERNAL_STORAGE, WRITEEXTERNAL_STORAGE</td>
</tr>
</tbody>
</table>

As of API level 23, the following permissions are classified as **PROTECTION_NORMAL**:

- ACCESS_LOCATION_EXTRA_COMMANDS
- ACCESS_NETWORK_STATE
- ACCESS_NOTIFICATION_POLICY
- ACCESS_WIFI_STATE
- BLUETOOTH
- BLUETOOTH_ADMIN
- BROADCAST_STICKY
- CHANGE_NETWORK_STATE
- CHANGE_WIFI_MULTICAST_STATE
- CHANGE_WIFI_STATE
- DISABLE_KEYGUARD
- EXPAND_STATUS_BAR
- GET_PACKAGE_SIZE
- INSTALL_SHORTCUT
- INTERNET
- KILL_BACKGROUND_PROCESSES
- MODIFY_AUDIO_SETTINGS
- NFC
- READ_SYNC_SETTINGS
- READ_SYNC_STATS
- RECEIVE_BOOT_COMPLETED
- REORDER_TASKS
- REQUEST_IGNORE_BATTERY_OPTIMIZATIONS
- REQUEST_INSTALL_PACKAGES
- SET_ALARM
- SET_TIME_ZONE
- SET_WALLPAPER
- SET_WALLPAPER_HINTS
- TRANSMIT_IR
- UNINSTALL_SHORTCUT
- USE_FINGERPRINT
- VIBRATE
- WAKE_LOCK
- WRITE_SYNC_SETTINGS
Permission development practices to follow

• Always Check for Permissions
  – When the app needs to perform any action that requires a permission, it should first check whether it has that permission

• Handle Lack of Permissions Gracefully
  – If the app is not granted an appropriate permission, it should handle the failure cleanly. For example, if the permission is just needed for an added feature, the app can disable that feature. If the permission is essential for the app to function, the app might disable all its functionality and inform the user that they need to grant that permission

• Permissions are Revocable
  – Users can revoke an app's permissions at any time
Runtime permission steps overview

Source: http://code.tutsplus.com/articles/understanding-permissions-in-android-m--cms-24443
// Add "compile 'pub.devrel:easypermissions:1.0.0' to dependencies in your build.gradle file
public class MainActivity extends AppCompatActivity implements EasyPermissions.PermissionCallbacks {

private static final int RC_LOCATION_STORAGE_PERM = 124;
private static final int RC_SETTINGS_SCREEN = 125;

@AfterPermissionGranted(RC_LOCATION_STORAGE_PERM)
public void locationAndStoragesTask() {
    String[] perms = {Manifest.permission.ACCESS_FINE_LOCATION, Manifest.permission.WRITE_EXTERNAL_STORAGE};
    if (EasyPermissions.hasPermissions(this, perms)) {
        Toast.makeText(this, "Permitted: Location and Storage!", Toast.LENGTH_LONG).show();
        getCurrentNetworkParametersCellid();
    } else {
        // Ask for both permissions
        EasyPermissions.requestPermissions(this,
            getString(R.string.rationale_location_storage),
            RC_LOCATION_STORAGE_PERM, perms);
    }
}

@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    // EasyPermissions handles the request result.
    EasyPermissions.onRequestPermissionsResult(requestCode, permissions, grantResults, this);
}

@Override
public void onPermissionsGranted(int requestCode, List<String> perms) {
    Log.d(TAG, "onPermissionsGranted:" + requestCode + ":" + perms.size());
}

@Override
public void onPermissionsDenied(int requestCode, List<String> perms) {
    Log.d(TAG, "onPermissionsDenied:" + requestCode + ":" + perms.size());
    // (Optional) Check whether the user denied any permissions and checked "NEVER ASK AGAIN."
    // This will display a dialog directing them to enable the permission in app settings.
    if (EasyPermissions.somePermissionPermanentlyDenied(this, perms)) {
        new AppSettingsDialog.Builder(this).build().show();
    }
}

// when accessing the resource handle exception
// in getCurrentNetworkParametersCellid()
try {
    location = (GsmCellLocation)
telephonyManager.getCellLocation();
} catch(SecurityException se){
    Toast.makeText(this, "SecurityException! " + se.getMessage(), Toast.LENGTH_LONG).show();
}
The official API is heavily coupled with the Activity class. Dexter frees your permission code from your activities and lets you write that logic anywhere you want.

https://github.com/Karumi/Dexter
Hotchemi

https://github.com/hotchemi/PermissionsDispatcher

• Provides a simple annotation-based API to handle runtime permissions

```java
@RuntimePermissions
class MainActivity extends AppCompatActivity {

    @NeedsPermission(Manifest.permission.CAMERA)
    public void showCamera() {
        getSupportFragmentManager().beginTransaction()
            .replace(R.id.sample_content_fragment, CameraPreviewFragment.newInstance())
            .addToBackStack("camera")
            .commitAllowingStateLoss();
    }

    @OnShowRationale(Manifest.permission.CAMERA)
    public void showRationaleForCamera(PermissionRequest request) {
        new AlertDialog.Builder(this)
            .setMessage(R.string.permission_camera_rationale)
            .setPositiveButton(R.string.button_allow, (dialog, button) -> request.proceed())
            .setNegativeButton(R.string.button_deny, (dialog, button) -> request.cancel())
            .show();
    }

    @OnPermissionDenied(Manifest.permission.CAMERA)
    public void showDeniedForCamera() {
        Toast.makeText(this, R.string.permission_camera_denied, Toast.LENGTH_SHORT).show();
    }

    @OnNeverAskAgain(Manifest.permission.CAMERA)
    public void showNeverAskForCamera() {
        Toast.makeText(this, R.string.permission_camera_neverask, Toast.LENGTH_SHORT).show();
    }
}
```

NOTE: Annotated methods must NOT be private!
Runtime permission steps explained
(Doing it the API way)

Step 1: check the platform – one line of code
Step 2: check the permission status - checkSelfPermission()
http://goo.gl/T7vE7b
Step 3: explain the permission - shouldShowRequestPermissionRationale()
http://goo.gl/bFyfVj
Step 4: request the permission - requestPermissions()  http://goo.gl/yNuizg
Step 5: handle the response - onRequestPermissionsResult() callback

Source: https://plus.google.com/+JoannaGSmith/posts/h4DuTT7tDKn
Check if the app has the needed permission (in this case: android.permission-group.LOCATION)

```java
// the Mapsproject Android Studio example uses the new permission model with real-time permissions
private static final int REQUEST_LOCATION = 10;
private static final String TAG = MainActivity.class.getSimpleName();
private static String[] PERMISSIONS_LOCATION = {Manifest.permission.ACCESS_COARSE_LOCATION,
                                              Manifest.permission.ACCESS_FINE_LOCATION};
// use the LocationManager class to obtain GPS locations
private LocationManager mLM;

public class MainActivity extends AppCompatActivity{
    ...
    /** Initialize location listener */
    private void initLocation() {
        // use the LocationManager class to obtain GPS locations
        mLM = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
        Log.d(TAG, "Checking permissions for Location...");

        // Check if the Location permission are already available
        if (ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) !=
                                        PackageManager.PERMISSION_GRANTED
                || ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_COARSE_LOCATION) !=
                                        PackageManager.PERMISSION_GRANTED) {
            // Location permissions has not been granted
            requestLocationPermission();
        } else {
            // Location permissions is already available, show the location
            Log.d(TAG, "Location permissions has already been granted. Requesting location!");
            mLM.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1000, 0, MyLocationListener);
        }
    }
    ...
```
_requestLocation permission

/** Request location permissions */
private void requestLocationPermission() {
    Log.d(TAG, "Location permissions has NOT been granted. Requesting permissions.");

    if (ActivityCompat.shouldShowRequestPermissionRationale(this,
        Manifest.permission.ACCESS_FINE_LOCATION)
        || ActivityCompat.shouldShowRequestPermissionRationale(this,
        Manifest.permission.ACCESS_COARSE_LOCATION)) {
        // Provide an additional rationale to the user if the permission was not granted
        // and the user would benefit from additional context for the use of the permission.
        // For example, if the request has been denied previously.
        Log.d(TAG, "Displaying location permission rationale to provide additional context");

        // Display a SnackBar with an explanation and a button to trigger the request
        Snackbar.make(mLayout, R.string.permission_location_rationale,
            Snackbar.LENGTH_INDEFINITE).setAction(R.string.ok, new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                ActivityCompat.requestPermissions(MainActivity.this, PERMISSIONS_LOCATION,
                    REQUEST_LOCATION);
            }
        }).show();
    }
    else {
        // Location permissions have not been granted yet. Request them directly.
        ActivityCompat.requestPermissions(this, PERMISSIONS_LOCATION, REQUEST_LOCATION);
    }
}
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults)
{
    if (requestCode == REQUEST_LOCATION) {
        Log.d(TAG, "Received response for location permissions request!");

        // We have requested multiple permissions for location, so all of them need to be checked.
        if (PermissionUtil.verifyPermissions(grantResults)) {
            // All required permissions have been granted, display location.
            Snackbar.make(mLayout, R.string.permission_available_location, Snackbar.LENGTH_SHORT)
                .show();
        }
    } else {
        Log.d(TAG, "Location permissions were NOT granted.");
        Snackbar.make(mLayout, R.string.permissions_not_granted, Snackbar.LENGTH_SHORT)
            .show();
    }
}
else {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
}
}
PermissionUtil class

/**
 * Utility class that wraps access to the runtime permissions API in M and provides basic helper methods.
 */

public abstract class PermissionUtil {

/**
 * Check that all given permissions have been granted by verifying that each entry in the
 * given array is of the value {@link PackageManager#PERMISSION_GRANTED}.
 *
 * @see Activity#onRequestPermissionsResult(int, String[], int[])
 */

public static boolean verifyPermissions(int[] grantResults) {

// At least one result must be checked.
if (grantResults.length < 1) {
    return false;
}

// Verify that each required permission has been granted, otherwise return false.
for (int result : grantResults) {
    if (result != PackageManager.PERMISSION_GRANTED) {
        return false;
    }
}

return true;
}

/**
 * Full example code
 * http://users.du.se/~hjo/cs/common/androidexamples/Mapsproject.7z
 * https://developer.android.com/about/versions/marshmallow/samples.html > RuntimePermissions
 */