Table of Contents

1. Pending Intent Mutability

Change content: Android 12 requires explicit specification of whether PendingIntent is mutable, FLAG_IMMUTABLE or FLAG_MUTABLE must be set

Reference link: https://developer.android.com/about/versions/12/behavior-changes-12#pending-intent-mutability

Code example:

```
// Create PendingIntent with explicit immutability specification
private PendingIntent createAlarmPendingIntent() {
    Intent intent = new Intent(this, AlarmReceiver.class);
    intent.setAction("com.example.app.ALARM_ACTION");
```

```
    // Explicitly specify FLAG_IMMUTABLE, indicating this PendingIntent cannot be modified
    return PendingIntent.getBroadcast(
            this, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT | PendingIntent.FLAG_IMMUTABLE);
}

// For PendingIntent that needs to be mutable (e.g., geofencing), use FLAG_MUTABLE
private PendingIntent createGeofencePendingIntent() {
    Intent intent = new Intent(this, GeofenceTransitionsIntentService.class);

    // Explicitly specify FLAG_MUTABLE, allowing system to modify data in this Intent
    return PendingIntent.getService(
            this, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT | PendingIntent.FLAG_MUTABLE);
}

// Create notification
private void showNotification() {
    Intent intent = new Intent(this, MainActivity.class);
    // Explicitly specify immutability, most notifications don't need mutability
    PendingIntent pendingIntent = PendingIntent.getActivity(
            this, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT | PendingIntent.FLAG_IMMUTABLE);

    NotificationCompat.Builder builder = new NotificationCompat.Builder(this, "channel_id")
            .setContentTitle("Notification Title")
            .setContentText("Notification Content")
            .setSmallIcon(R.drawable.ic_notification)
            .setContentIntent(pendingIntent);

    NotificationManager notificationManager =
            (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        NotificationChannel channel = new NotificationChannel(
                "channel_id", "Channel Name", NotificationManager.IMPORTANCE_DEFAULT);
        notificationManager.createNotificationChannel(channel);
    }
```

```
    notificationManager.notify(1, builder.build());
}

// DIFFERENCE: For compatibility with different versions, use helper method
private int getPendingIntentImmutableFlag() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        return PendingIntent.FLAG_UPDATE_CURRENT | PendingIntent.FLAG_IMMUTABLE;
      // DIFFERENCE: Added FLAG_IMMUTABLE
    } else {
        return PendingIntent.FLAG_UPDATE_CURRENT; // DIFFERENCE: No FLAG_IMMUTABLE
    }
}
```

2. Notification System Changes

Change point: Notification templates and styles update

Android 12 has updated the notification system visually and functionally to align with the Material You design language.

```
// Notification implementation for Android 10 and Android 12
private void showNotification() {
    NotificationCompat.Builder builder = new NotificationCompat.Builder(context, CHANNEL_ID)
        .setSmallIcon(R.drawable.notification_icon)
        .setContentTitle("Notification Title")
        .setContentText("Notification Content")
        .setPriority(NotificationCompat.PRIORITY_DEFAULT);

    // DIFFERENCE: Android 12 supports richer styles
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) { // Android 12 is API level 31
        builder.setStyle(new NotificationCompat.DecoratedCustomViewStyle());
        // On Android 12, notification visual style will automatically adapt to Material You design
    }
```

```
    NotificationManagerCompat notificationManager = NotificationManagerCompat.from(context);
    notificationManager.notify(notificationId, builder.build());
}
```

Main changes:

1. Visual update: Notification UI updated, adopting Material You design language
2. Larger touch targets: Notification action buttons are larger, easier to tap
3. Adaptive colors: Notification colors can adapt to system theme
4. Improved media controls: Media notifications get new design and functionality
5. Notification animations: New expand and collapse animations

3. System UI Flag Deprecation

Change content:

- SYSTEM_UI_FLAG_* constants are deprecated
- Should use WindowInsetsController instead

Reference link: https://developer.android.com/sdk/api_diff/30/changes

```
// Method to manage system UI: hide bars, check visibility, set theme, and listen for changes
private void manageSystemUI() {
    View decorView = getWindow().getDecorView();
    WindowInsetsController controller = decorView.getWindowInsetsController();

    if (controller != null) {
        // DIFFERENCE: HIDE SYSTEM BARS
        controller.hide(WindowInsets.Type.systemBars());

        // DIFFERENCE: SET IMMERSIVE MODE BEHAVIOR
```

```
        controller.setSystemBarsBehavior(
            WindowInsetsController.BEHAVIOR_SHOW_TRANSIENT_BARS_BY_SWIPE
        );

        // DIFFERENCE: SET STATUS BAR TO LIGHT THEME
        controller.setSystemBarsAppearance(
            WindowInsetsController.APPEARANCE_LIGHT_STATUS_BARS,
            WindowInsetsController.APPEARANCE_LIGHT_STATUS_BARS
        );
    }

    // DIFFERENCE: CHECK SYSTEM UI VISIBILITY
    WindowInsets insets = decorView.getRootWindowInsets();
    boolean isStatusBarVisible = insets.isVisible(WindowInsets.Type.statusBars());
    boolean isNavBarVisible = insets.isVisible(WindowInsets.Type.navigationBars());

    // DIFFERENCE: LISTEN FOR SYSTEM UI VISIBILITY CHANGES
    decorView.setOnApplyWindowInsetsListener(new View.OnApplyWindowInsetsListener() {
        @Override
        public WindowInsets onApplyWindowInsets(View v, WindowInsets insets) {
            boolean isStatusBarVisible = insets.isVisible(WindowInsets.Type.statusBars());
            // Handle visibility changes
            return v.onApplyWindowInsets(insets);
        }
    });
}
```

4. Toast Notification Restrictions

Change content:

- Block custom Toast from background applications
- Add new addCallback() method to monitor Toast show and hide

```
// Before Android 11: Custom Toasts in Foreground and Background
// Create custom Toast
Toast toast = new Toast(context);

// Create custom view
LinearLayout layout = new LinearLayout(context);
layout.setBackgroundColor(Color.RED);
TextView textView = new TextView(context);
textView.setText("This is a custom Toast");
textView.setTextColor(Color.WHITE);
layout.addView(textView);

// Set custom view
toast.setView(layout);
toast.setDuration(Toast.LENGTH_LONG);

// Show Toast - works normally in both foreground and background
toast.show();

// DIFFERENCE: Cannot monitor Toast show and hide
// No callback mechanism

// Android 11 and Later: Changes and Restrictions
// 1. Background Custom Toast Restrictions
// In background service:
Toast toast = new Toast(context);
LinearLayout layout = new LinearLayout(context);
// ... set up custom view ...
toast.setView(layout);
toast.show();

// Result: Toast won't show, system will log in logcat:
// W/NotificationService: Blocking custom toast from package <package> due to package not in the
foreground

// 2. Background Text Toast Still Allowed
```

```
// In background service:
// This standard text Toast can still be shown in background
Toast.makeText(context, "This is a standard text Toast", Toast.LENGTH_LONG).show();

// 3. New Toast Callback API
// Create Toast
Toast toast = Toast.makeText(context, "Toast with callback", Toast.LENGTH_LONG);

// Add callback to monitor Toast show and hide
toast.addCallback(new Toast.Callback() {
    @Override
    public void onToastShown() {
        Log.d("ToastDemo", "Toast has been shown");
    }

    @Override
    public void onToastHidden() {
        Log.d("ToastDemo", "Toast has been hidden");
    }
});

toast.show();
// DIFFERENCE: Android 11 added the addCallback() method, allowing applications to monitor Toast show and hide events.

// 4. Text Toast API Changes
// Before Android 11
Toast toast = Toast.makeText(context, "Text Toast", Toast.LENGTH_LONG);
toast.setGravity(Gravity.TOP | Gravity.CENTER_HORIZONTAL, 0, 100);
View view = toast.getView(); // Returns actual view
float horizontalMargin = toast.getHorizontalMargin(); // Returns actual value

// Android 11 and later (for applications targeting SDK version >= 30)
Toast toast = Toast.makeText(context, "Text Toast", Toast.LENGTH_LONG);
toast.setGravity(Gravity.TOP | Gravity.CENTER_HORIZONTAL, 0, 100); // No effect
View view = toast.getView(); // Returns null
```

```
float horizontalMargin = toast.getHorizontalMargin(); // Return value doesn't reflect actual value
```

5. Device Identifier Restrictions

Change name: Device Identifier Restrictions

Change content:

Android 12 further restricts applications' ability to access device identifiers, enhancing user privacy protection.

Reference link:

https://developer.android.com/about/versions/12/behavior-changes-all?hl=en#device-identifiers

```
// Method to access device identifiers, highlighting differences between Android 10 and Android 12
private void getDeviceIdentifiers() {
    // DIFFERENCE: IMEI Access
    if (Build.VERSION.SDK_INT < Build.VERSION_CODES.S) {
        // Android 10: Access IMEI (requires READ_PHONE_STATE permission)
        TelephonyManager telephonyManager = (TelephonyManager)
getSystemService(Context.TELEPHONY_SERVICE);
        String imei = telephonyManager.getImei();
    } else {
        // Android 12: IMEI access is restricted, requires special permission
        if (checkSelfPermission(Manifest.permission.READ_PHONE_STATE) ==
PackageManager.PERMISSION_GRANTED) {
            TelephonyManager telephonyManager = (TelephonyManager)
getSystemService(Context.TELEPHONY_SERVICE);
            // Getting IMEI requires special permission
            // String imei = telephonyManager.getImei(); // May not be accessible
        }
    }

    // DIFFERENCE: MAC Address Access
    if (Build.VERSION.SDK_INT < Build.VERSION_CODES.S) {
```

```
        // Android 10: Get MAC address
        WifiManager wifiManager = (WifiManager)
getApplicationContext().getSystemService(Context.WIFI_SERVICE);
        WifiInfo wifiInfo = wifiManager.getConnectionInfo();
        String macAddress = wifiInfo.getMacAddress();
    } else {
        // Android 12: MAC address access is restricted
        // Cannot get real MAC address, use alternative methods
    }

    // Android ID Access
    String androidId = Settings.Secure.getString(getContentResolver(), Settings.Secure.ANDROID_ID);

    // RECOMMENDATION: Use more modern identification methods
    String appSpecificId = UUID.randomUUID().toString();
    // Store in SharedPreferences
    SharedPreferences prefs = getSharedPreferences("app_prefs", MODE_PRIVATE);
    if (!prefs.contains("app_id")) {
        prefs.edit().putString("app_id", appSpecificId).apply();
    }
    String storedId = prefs.getString("app_id", appSpecificId);
}
```

6. Scoped Storage Enforcement

Change content:

Android 10 (Scoped Storage opt-out possible)

Android 10 introduced scoped storage, but applications could opt out by adding requestLegacyExternalStorage attribute in the manifest file:

```
<manifest ... >
    <application
        android:requestLegacyExternalStorage="true"
```

```
    ... >
    ...
  </application>
</manifest>
```

With this flag set, applications could still use traditional storage access mode.

Android 11 (Enforced Scoped Storage)

In Android 11, all applications targeting API 30 must use scoped storage regardless of the requestLegacyExternalStorage setting. Here are the new access modes:

1. App-specific directory access

```
// Access app-specific directory (no storage permission needed)
File appSpecificExternalDir = getExternalFilesDir(null);
File myFile = new File(appSpecificExternalDir, "data.txt");

try {
  FileOutputStream fos = new FileOutputStream(myFile);
  fos.write("Hello World".getBytes());
  fos.close();
  // Successfully wrote file to app-specific directory
} catch (IOException e) {
  e.printStackTrace();
}

// These files will be deleted when the app is uninstalled
```

2. Media file access (using MediaStore API)

```
// Save image to shared media storage
ContentValues values = new ContentValues();
values.put(MediaStore.Images.Media.DISPLAY_NAME, "my_image.jpg");
```

```
values.put(MediaStore.Images.Media.MIME_TYPE, "image/jpeg");
values.put(MediaStore.Images.Media.RELATIVE_PATH, "Pictures/MyApp");

ContentResolver resolver = getContentResolver();
Uri imageUri = resolver.insert(MediaStore.Images.Media.EXTERNAL_CONTENT_URI, values);

try {
    OutputStream os = resolver.openOutputStream(imageUri);
    // Write image data to output stream
    Bitmap bitmap = getBitmapFromSomewhere();
    bitmap.compress(Bitmap.CompressFormat.JPEG, 90, os);
    os.close();
} catch (IOException e) {
    e.printStackTrace();
}

// Query self-created media files (requires READ_EXTERNAL_STORAGE permission)
String[] projection = {
    MediaStore.Images.Media._ID,
    MediaStore.Images.Media.DISPLAY_NAME
};

String selection = MediaStore.Images.Media.RELATIVE_PATH + " LIKE ?";
String[] selectionArgs = new String[]{"Pictures/MyApp%"};

Cursor cursor = resolver.query(
    MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
    projection,
    selection,
    selectionArgs,
    null
);

// Process query results
while (cursor.moveToNext()) {
    // Get image information
```

```
}
cursor.close();
```

3. Access other apps' files (no longer allowed)

```
// In Android 11, this code will fail
File externalDir = Environment.getExternalStorageDirectory();
File otherAppFile = new File(externalDir, "OtherApp/data.txt");
try {
    FileInputStream fis = new FileInputStream(otherAppFile);
    // Will throw exception, cannot access other apps' files
    fis.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

4. Use Storage Access Framework (SAF) to access files

```
// Launch file picker for user to select file
Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT);
intent.addCategory(Intent.CATEGORY_OPENABLE);
intent.setType("*/*");
startActivityForResult(intent, REQUEST_CODE);

// Handle selected file in onActivityResult
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == REQUEST_CODE && resultCode == RESULT_OK) {
        if (data != null) {
            Uri uri = data.getData();
            try {
                InputStream is = getContentResolver().openInputStream(uri);
                // Read file content
                is.close();
```

```
            // Keep long-term access permission (if needed)
            getContentResolver().takePersistableUriPermission(uri,
                Intent.FLAG_GRANT_READ_URI_PERMISSION);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
  }
}
```

5. Access specific directories (using SAF)

```
// Launch directory picker
Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT_TREE);
startActivityForResult(intent, REQUEST_DIRECTORY_CODE);

// Handle selected directory in onActivityResult
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
  super.onActivityResult(requestCode, resultCode, data);
  if (requestCode == REQUEST_DIRECTORY_CODE && resultCode == RESULT_OK) {
    if (data != null) {
      Uri treeUri = data.getData();

      // Keep long-term access permission
      getContentResolver().takePersistableUriPermission(treeUri,
        Intent.FLAG_GRANT_READ_URI_PERMISSION |
        Intent.FLAG_GRANT_WRITE_URI_PERMISSION);

      // Use DocumentFile to access directory contents
      DocumentFile pickedDir = DocumentFile.fromTreeUri(this, treeUri);
      DocumentFile[] files = pickedDir.listFiles();
      for (DocumentFile file : files) {
        Log.d("Files", "Found: " + file.getName());
```

```
        }

        // Create new file in selected directory
        DocumentFile newFile = pickedDir.createFile("text/plain", "newfile.txt");
        try {
            OutputStream os = getContentResolver().openOutputStream(newFile.getUri());
            os.write("Hello World".getBytes());
            os.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
  }
}
```

7. Foreground Service Launch Restrictions

Change content: Android 12 restricts the ability to start foreground services from the background, must use specific methods to trigger

Reference link: https://developer.android.com/about/versions/12/behavior-changes-12#foreground-service-launch-restrictions

Code example:

```
// BackgroundWorker class for starting services
public class BackgroundWorker extends Worker {
  @NonNull
  @Override
  public Result doWork() {
    Context context = getApplicationContext();

    // DIFFERENCE: Starting Foreground Service
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
      // Android 12: Cannot start foreground service directly from background
```

```
            // Use notification to prompt user action
            Intent serviceIntent = new Intent(context, MyForegroundService.class);
            PendingIntent pendingIntent = PendingIntent.getService(
                    context, 0, serviceIntent, PendingIntent.FLAG_IMMUTABLE);

            NotificationCompat.Builder builder = new NotificationCompat.Builder(context, "channel_id")
                    .setContentTitle("Service Start Required")
                    .setContentText("Click this notification to start necessary service")
                    .setSmallIcon(R.drawable.ic_notification)
                    .setContentIntent(pendingIntent)
                    .setAutoCancel(true);

            NotificationManager notificationManager =
                    (NotificationManager) context.getSystemService(Context.NOTIFICATION_SERVICE);

            // Ensure notification channel is created
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
                NotificationChannel channel = new NotificationChannel(
                        "channel_id", "Channel Name", NotificationManager.IMPORTANCE_DEFAULT);
                notificationManager.createNotificationChannel(channel);
            }

            notificationManager.notify(2, builder.build());
        } else {
            // Android 11 and below: Can start foreground service directly
            Intent serviceIntent = new Intent(context, MyForegroundService.class);
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
                context.startForegroundService(serviceIntent);
            } else {
                context.startService(serviceIntent);
            }
        }

        return Result.success();
    }
}
```

```
// MyForegroundService class for foreground service implementation
public class MyForegroundService extends Service {
    @Override
    public void onCreate() {
        super.onCreate();

        // Create notification channel for foreground service
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            NotificationChannel channel = new NotificationChannel(
                    "channel_id", "Channel Name", NotificationManager.IMPORTANCE_DEFAULT);
            NotificationManager notificationManager = getSystemService(NotificationManager.class);
            notificationManager.createNotificationChannel(channel);
        }

        Notification notification = new NotificationCompat.Builder(this, "channel_id")
                .setContentTitle("Foreground Service")
                .setContentText("Service is running")
                .setSmallIcon(R.drawable.ic_notification)
                .build();

        startForeground(1, notification);
    }
}

// ForegroundWorker class using WorkManager for foreground tasks
public class ForegroundWorker extends Worker {
    public ForegroundWorker(@NonNull Context context, @NonNull WorkerParameters params) {
        super(context, params);
    }

    @NonNull
    @Override
    public Result doWork() {
        // Execute work that needs to be done in foreground service
        // ...
```

```
    return Result.success();
}

// Use WorkManager to start foreground service (recommended method)
public static void enqueueForegroundWork(Context context) {
    Constraints constraints = new Constraints.Builder()
            .setRequiredNetworkType(NetworkType.CONNECTED)
            .build();

    // Create foreground info
    ForegroundInfo foregroundInfo = createForegroundInfo(context);

    // Create one-time work request
    OneTimeWorkRequest workRequest = new OneTimeWorkRequest.Builder(ForegroundWorker.class)
            .setConstraints(constraints)
            .build();

    // Enqueue work request
    WorkManager.getInstance(context)
            .enqueueUniqueWork(
                    "foreground_work",
                    ExistingWorkPolicy.REPLACE,
                    workRequest);

    // Set as foreground service
    WorkManager.getInstance(context).getWorkInfoByIdLiveData(workRequest.getId())
            .observeForever(workInfo -> {
                if (workInfo != null && workInfo.getState() == WorkInfo.State.RUNNING) {
                    WorkManager.getInstance(context).setForegroundAsync(
                            workRequest.getId(), foregroundInfo);
                }
            });
}

// Create foreground info needed for foreground service
```

```
    private static ForegroundInfo createForegroundInfo(Context context) {
        // Create notification channel
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            NotificationChannel channel = new NotificationChannel(
                    "foreground_channel",
                    "Foreground Work",
                    NotificationManager.IMPORTANCE_LOW);

            NotificationManager notificationManager =
                    (NotificationManager) context.getSystemService(Context.NOTIFICATION_SERVICE);
            notificationManager.createNotificationChannel(channel);
        }

        // Create notification
        Notification notification = new NotificationCompat.Builder(context, "foreground_channel")
                .setContentTitle("Foreground Work")
                .setContentText("Work in progress...")
                .setSmallIcon(R.drawable.ic_notification)
                .setOngoing(true)
                .build();

        return new ForegroundInfo(NOTIFICATION_ID, notification);
    }
}
```

8. Splash Screen Changes

Change point: Splash Screen API

Android 12 introduces the new SplashScreen API, which is a major change providing a unified launch experience for all applications.

```
// Android 10 Implementation: Manual Splash Screen
public class SplashActivity extends AppCompatActivity {
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_splash);

        // DIFFERENCE: Manual delay for splash screen
        new Handler().postDelayed(() -> {
            startActivity(new Intent(SplashActivity.this, MainActivity.class));
            finish();
        }, 2000);
    }
}

// AndroidManifest.xml for Android 10
<activity
    android:name=".SplashActivity"
    android:theme="@style/SplashTheme">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

// Android 12 Implementation: Using SplashScreen API
// Configure in themes.xml
<style name="Theme.App" parent="Theme.MaterialComponents.DayNight.NoActionBar">
    <!-- DIFFERENCE: Splash screen configuration in theme -->
    <item name="android:windowSplashScreenBackground">@color/splash_background</item>
    <item name="android:windowSplashScreenAnimatedIcon">@drawable/splash_icon</item>
    <item name="android:windowSplashScreenAnimationDuration">1000</item>
    <item name="android:windowSplashScreenBrandingImage">@drawable/branding_image</item>
</style>

// MainActivity for Android 12
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);

        // DIFFERENCE: Use SplashScreen API on Android 12
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
            SplashScreen splashScreen = getSplashScreen();

            // Extend splash screen display time until data loading is complete
            splashScreen.setOnExitAnimationListener(splashScreenView -> {
                // Execute custom exit animation
                ObjectAnimator fadeOut = ObjectAnimator.ofFloat(
                    splashScreenView.getView(),
                    View.ALPHA,
                    1f,
                    0f
                );
                fadeOut.setDuration(500);
                fadeOut.addListener(new AnimatorListenerAdapter() {
                    @Override
                    public void onAnimationEnd(Animator animation) {
                        splashScreenView.remove();
                    }
                });
                fadeOut.start();
            });
        }

        setContentView(R.layout.activity_main);
    }
}
```

Main changes:

1. Automatic splash screen generation: Android 12 automatically generates splash screens for all apps, no need for developers to manually create dedicated splash Activities

2. Unified experience: Provides consistent transition animation from app icon to app content
3. Declarative configuration: Configure splash screen appearance through theme attributes
4. Programmatic control: Can control splash screen duration and exit animation through code
5. Brand showcase: Support displaying brand image at bottom of splash screen
6. Backward compatibility: Through Core Splashscreen library can be backward compatible to API level 23

9. Custom Intent Filter Verification

Change content: Android 12 has stricter verification of intent filters for inter-app interactions

Reference link: https://developer.android.com/about/versions/12/behavior-changes-12#custom-intent-filter-verification

Code example:

```
<!-- AndroidManifest.xml -->
<!-- Android 10: Declare intent filters -->
<activity android:name=".CustomActivity">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE" />
        <data android:scheme="custom" android:host="example.com" />
    </intent-filter>
</activity>

<!-- Android 12: Declare intent filters with export state -->
<activity
    android:name=".CustomActivity"
    android:exported="true"> <!-- DIFFERENCE: Must declare exported state -->
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
```

```
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE" />
        <data android:scheme="custom" android:host="example.com" />
    </intent-filter>
</activity>


// Android 10: Send custom intent
private void sendCustomIntent() {
    Intent intent = new Intent(Intent.ACTION_VIEW);
    intent.setData(Uri.parse("custom://example.com/path"));
    startActivity(intent);
}

// Android 12: Send custom intent safely
private void sendCustomIntentSafely() {
    Intent intent = new Intent(Intent.ACTION_VIEW);
    intent.setData(Uri.parse("custom://example.com/path"));

    // DIFFERENCE: Verify receiver exists before sending intent
    PackageManager packageManager = getPackageManager();
    List<ResolveInfo> activities = packageManager.queryIntentActivities(
            intent, PackageManager.MATCH_DEFAULT_ONLY);

    if (activities.size() > 0) {
        // Found application that can handle this intent
        startActivity(intent);
    } else {
        // No application can handle this intent
        Toast.makeText(this, "No application can handle this action", Toast.LENGTH_SHORT).show();

        // Provide fallback option
        Intent browserIntent = new Intent(Intent.ACTION_VIEW);
        browserIntent.setData(Uri.parse("https://example.com/path"));
        startActivity(browserIntent);
    }
```

```
}
```

10. AppSearch and WebView Changes

Change name: AppSearch and WebView Changes

Change content:

Android 12 introduced the AppSearch API, providing more powerful in-app search functionality, while also making several improvements to WebView.

Reference link:

https://developer.android.com/about/versions/12/features#appsearch

https://developer.android.com/about/versions/12/behavior-changes-all?hl=en#webview

Example code:

```
// Android 10: In-app search using SQLite
public class SearchManager {
    private SQLiteDatabase db;

    public SearchManager(Context context) {
        SearchDbHelper dbHelper = new SearchDbHelper(context);
        db = dbHelper.getWritableDatabase();
    }

    public List<SearchResult> search(String query) {
        List<SearchResult> results = new ArrayList<>();
        Cursor cursor = db.query(
            "search_table",
            new String[]{"id", "title", "content"},
            "title LIKE ? OR content LIKE ?",
            new String[]{"%" + query + "%", "%" + query + "%"},
            null, null, null);
```

```
        while (cursor.moveToNext()) {
            SearchResult result = new SearchResult(
                cursor.getLong(0),
                cursor.getString(1),
                cursor.getString(2)
            );
            results.add(result);
        }
        cursor.close();
        return results;
    }
}

// Android 12: In-app search using AppSearch API
@RequiresApi(api = Build.VERSION_CODES.S)
public class AppSearchManager {
    private AppSearchSession searchSession;

    public AppSearchManager(Context context) {
        // Initialize AppSearch
        AppSearchManager appSearchManager = (AppSearchManager)
context.getSystemService(Context.APP_SEARCH_SERVICE);
        appSearchManager.createSearchSession(
            new SearchContext.Builder(context, "database_name").build())
            .addOnSuccessListener(session -> {
                searchSession = session;
            });
    }

    public void search(String query, SearchResultsCallback callback) {
        if (searchSession == null) return;

        SearchSpec searchSpec = new SearchSpec.Builder()
            .setTermMatch(SearchSpec.TERM_MATCH_PREFIX)
            .build();
```

```
        searchSession.search(
            query,
            searchSpec,
            new Executor() {
                @Override
                public void execute(Runnable command) {
                    new Handler(Looper.getMainLooper()).post(command);
                }
            },
            new SearchResultsCallback() {
                @Override
                public void onResult(SearchResults results) {
                    List<SearchResult> searchResults = new ArrayList<>();
                    for (SearchResult result : results.getResults()) {
                        // Process search results
                        searchResults.add(result);
                    }
                    callback.onResult(results);
                }
            });
    }
}

// WebView usage for both Android 10 and Android 12
WebView webView = findViewById(R.id.webview);
WebSettings settings = webView.getSettings();
settings.setJavaScriptEnabled(true);
webView.loadUrl("https://example.com");

// Android 12: WebView improvements
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
    // Set WebView render priority
    settings.setWebViewRenderProcessGoneListener((webView1, renderer) -> {
        // Handle render process crash
        return true; // Return true indicates application has handled the crash
```

```
    });

    // Use new Cookie management API
    CookieManager.getInstance().setAcceptThirdPartyCookies(webView, false);
}
```

11. Approximate Location Permission

Change point: Location permission granularity

Android 12 introduced approximate location permission, allowing users to grant only approximate location instead of precise location.

```
// Location permission request for Android 10 and Android 12
private void requestLocationPermission() {
    // DIFFERENCE: Android 12 allows users to choose between precise and approximate location
    String[] permissions = new String[]{
        Manifest.permission.ACCESS_FINE_LOCATION
    };

    requestPermissions(permissions, REQUEST_LOCATION_PERMISSION);
}

// Check if has precise location permission
private boolean hasExactLocationPermission() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
        // Android 12 and above needs to check if has precise location permission
        return checkSelfPermission(Manifest.permission.ACCESS_FINE_LOCATION) ==
                PackageManager.PERMISSION_GRANTED;
    } else {
        // Old versions with any location permission is precise
        return checkSelfPermission(Manifest.permission.ACCESS_COARSE_LOCATION) ==
                PackageManager.PERMISSION_GRANTED;
    }
}
```

```
// Get location for Android 10 and Android 12
private void getLocation() {
   boolean hasCoarseLocation = checkSelfPermission(Manifest.permission.ACCESS_COARSE_LOCATION)
==
         PackageManager.PERMISSION_GRANTED;

   if (hasCoarseLocation) {
      LocationManager locationManager =
         (LocationManager) getSystemService(Context.LOCATION_SERVICE);

      // DIFFERENCE: Choose provider based on permission type
      String provider = hasExactLocationPermission() ?
            LocationManager.GPS_PROVIDER : LocationManager.NETWORK_PROVIDER;

      locationManager.requestLocationUpdates(provider, 0, 0, locationListener);
   }
}
```

Main changes:

1. Approximate location option: Users can choose to grant only approximate location permission
2. Permission dialog update: Location permission dialog adds precise/approximate options
3. Permission check: Applications need to check if precise location permission is granted
4. Fallback handling: Applications need to handle cases with only approximate location
5. Location accuracy: Approximate location accuracy is about 3 kilometers range

12. Exact Alarm Permission

Change content: Android 12 requires SCHEDULE_EXACT_ALARM permission to set exact alarms

Reference link: https://developer.android.com/about/versions/12/behavior-changes-12#exact-alarm-permission

Code example:

```
<!-- AndroidManifest.xml -->
<!-- DIFFERENCE: Add permission for exact alarms in Android 12 -->
<uses-permission android:name="android.permission.SCHEDULE_EXACT_ALARM" />


// Schedule exact alarm for Android 10 and Android 12
private void scheduleExactAlarm() {
    AlarmManager alarmManager = (AlarmManager) getSystemService(Context.ALARM_SERVICE);

    // DIFFERENCE: Check permission for exact alarms in Android 12
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
        if (!alarmManager.canScheduleExactAlarms()) {
            // No permission, guide user to grant permission
            Intent intent = new Intent(Settings.ACTION_REQUEST_SCHEDULE_EXACT_ALARM);
            intent.setData(Uri.parse("package:" + getPackageName()));
            startActivity(intent);
            return;
        }
    }

    Intent intent = new Intent(this, AlarmReceiver.class);
    PendingIntent pendingIntent = PendingIntent.getBroadcast(
            this, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT | PendingIntent.FLAG_IMMUTABLE);

    // Set alarm time
    long triggerTimeMillis = System.currentTimeMillis() + 60 * 60 * 1000; // 1 hour later

    // Set exact alarm
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        alarmManager.setExactAndAllowWhileIdle(
                AlarmManager.RTC_WAKEUP, triggerTimeMillis, pendingIntent);
```

```
    } else if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT) {
        alarmManager.setExact(AlarmManager.RTC_WAKEUP, triggerTimeMillis, pendingIntent);
    } else {
        alarmManager.set(AlarmManager.RTC_WAKEUP, triggerTimeMillis, pendingIntent);
    }
}
```

13. Non-SDK Interface Restrictions

Change content: Android 12 further restricts access to non-SDK interfaces, more previously available non-public APIs are added to greylist or blacklist

Reference link: https://developer.android.com/about/versions/12/behavior-changes-12#non-sdk-interfaces

Code example:

```
// Android 10: Accessing non-SDK interfaces using reflection
private void accessHiddenApis() {
    try {
        // DIFFERENCE: Access hidden method of ActivityManager
        Class<?> activityManagerClass = Class.forName("android.app.ActivityManager");
        Method getDefaultMethod = activityManagerClass.getDeclaredMethod("getDefault");
        getDefaultMethod.setAccessible(true);
        Object activityManagerInstance = getDefaultMethod.invoke(null);

        // Access hidden field
        Field mConfigField = activityManagerClass.getDeclaredField("mConfiguration");
        mConfigField.setAccessible(true);
        Object config = mConfigField.get(activityManagerInstance);

        Log.d("HiddenAPI", "Successfully accessed hidden API: " + config);
    } catch (Exception e) {
        Log.e("HiddenAPI", "Failed to access hidden API", e);
    }
```

```
}

// Android 12: Using public APIs and handling non-SDK interfaces
private void usePublicApis() {
    // DIFFERENCE: Use public ActivityManager API
    ActivityManager activityManager = (ActivityManager) getSystemService(Context.ACTIVITY_SERVICE);

    // Get configuration information
    Configuration configuration = getResources().getConfiguration();

    Log.d("PublicAPI", "Using public API: " + configuration);

    // Enable StrictMode to detect non-SDK interface usage
    StrictMode.setVmPolicy(new StrictMode.VmPolicy.Builder()
            .detectNonSdkApiUsage()
            .penaltyLog()
            .build());

    // If really need to access non-SDK interfaces, check API availability before using reflection
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.P) {
        try {
            Method forNameMethod = Class.class.getDeclaredMethod("forName", String.class);
            Method getDeclaredMethodMethod = Class.class.getDeclaredMethod("getDeclaredMethod",
                    String.class, Class[].class);

            Class<?> vmRuntimeClass = (Class<?>) forNameMethod.invoke(null, "dalvik.system.VMRuntime");
            Method getRuntime = (Method) getDeclaredMethodMethod.invoke(vmRuntimeClass, "getRuntime",
                    null);
            Method setHiddenApiExemptions = (Method) getDeclaredMethodMethod.invoke(vmRuntimeClass,
                    "setHiddenApiExemptions", new Class[] { String[].class });

            Object vmRuntime = getRuntime.invoke(null);
            setHiddenApiExemptions.invoke(vmRuntime, new Object[] { new String[] { "L" } });
        } catch (Exception e) {
            Log.e("HiddenAPI", "Cannot enable hidden API access", e);
        }
```

```
    }
}
```

14. Intent Receiver Explicit Declaration

Change name: Intent Receiver Explicit Declaration

Change content:

Android 12 requires developers to explicitly declare the exported property of Intent receivers, enhancing application security.

Reference link:

https://developer.android.com/about/versions/12/behavior-changes-all?hl=en#receiver-exported

Example code:

```
<!-- AndroidManifest.xml -->
<!-- Android 10: Declare intent filters -->
<activity android:name=".CustomActivity">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE" />
        <data android:scheme="custom" android:host="example.com" />
    </intent-filter>
</activity>

<!-- Android 12: Declare intent filters with export state -->
<activity
    android:name=".CustomActivity"
    android:exported="true"> <!-- DIFFERENCE: Must declare exported state -->
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
```

```
        <category android:name="android.intent.category.BROWSABLE" />
        <data android:scheme="custom" android:host="example.com" />
    </intent-filter>
</activity>


// Android 10: Send custom intent
private void sendCustomIntent() {
    Intent intent = new Intent(Intent.ACTION_VIEW);
    intent.setData(Uri.parse("custom://example.com/path"));
    startActivity(intent);
}

// Android 12: Send custom intent safely
private void sendCustomIntentSafely() {
    Intent intent = new Intent(Intent.ACTION_VIEW);
    intent.setData(Uri.parse("custom://example.com/path"));

    // DIFFERENCE: Verify receiver exists before sending intent
    PackageManager packageManager = getPackageManager();
    List<ResolveInfo> activities = packageManager.queryIntentActivities(
            intent, PackageManager.MATCH_DEFAULT_ONLY);

    if (activities.size() > 0) {
        // Found application that can handle this intent
        startActivity(intent);
    } else {
        // No application can handle this intent
        Toast.makeText(this, "No application can handle this action", Toast.LENGTH_SHORT).show();

        // Provide fallback option
        Intent browserIntent = new Intent(Intent.ACTION_VIEW);
        browserIntent.setData(Uri.parse("https://example.com/path"));
        startActivity(browserIntent);
    }
```

```
}
```

15. App Hibernation

Change content: Android 12 introduces app hibernation feature, where apps that haven't been used for an extended period will enter hibernation state, permissions will be revoked, and cache will be cleared

Reference link: https://developer.android.com/about/versions/12/behavior-changes-12#app-hibernation

Code example:

```
// Android 10: Perform background operations without checking hibernation state
private void performBackgroundOperations() {
    // Unconditionally execute background operations
    syncData();
    prefetchContent();
    updateCache();
}

// Android 12: Perform background operations adaptively based on hibernation state
private void performBackgroundOperationsAdaptively() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
        AppHibernationManager hibernationManager = getSystemService(AppHibernationManager.class);

        if (hibernationManager != null &&
                !hibernationManager.isHibernatingForUser(getPackageName())) {
            // App is not in hibernation state, can perform operations
            syncData();
            prefetchContent();
            updateCache();
        } else {
            // App is in hibernation state, minimize operations
            Log.d("Hibernation", "App is in hibernation state, skipping non-essential operations");
```

```
        }
    } else {
        // Android 11 and below don't have hibernation feature
        performBackgroundOperations();
    }
}

// Schedule background tasks for both Android 10 and Android 12
private void scheduleBackgroundWork() {
    WorkManager workManager = WorkManager.getInstance(this);

    // Create periodic work request
    PeriodicWorkRequest workRequest = new PeriodicWorkRequest.Builder(
            SyncWorker.class,
            1, TimeUnit.HOURS)
            .build();

    workManager.enqueueUniquePeriodicWork(
            "sync_work",
            ExistingPeriodicWorkPolicy.REPLACE,
            workRequest);
}

// Android 12: Detect app usage patterns and adjust operations
private void adaptToUsagePatterns() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.P) {
        UsageStatsManager usageStatsManager =
                (UsageStatsManager) getSystemService(Context.USAGE_STATS_SERVICE);

        long now = System.currentTimeMillis();
        long dayAgo = now - (24 * 60 * 60 * 1000);

        // Get app usage stats for last day
        if (checkUsageStatsPermission()) {
            List<UsageStats> stats = usageStatsManager.queryUsageStats(
                    UsageStatsManager.INTERVAL_DAILY, dayAgo, now);
```

```
        // Check if has recent usage
        boolean hasRecentUsage = false;
        for (UsageStats usageStats : stats) {
            if (usageStats.getPackageName().equals(getPackageName()) &&
                    usageStats.getLastTimeUsed() > dayAgo) {
                hasRecentUsage = true;
                break;
            }
        }

        // Adjust cache size and sync frequency based on usage
        if (hasRecentUsage) {
            increaseCacheQuota();
            scheduleFrequentSync();
        } else {
            reduceCacheQuota();
            scheduleInfrequentSync();
        }
    }
  }
}

// Android 12: Handle app recovery from hibernation
@Override
protected void onResume() {
  super.onResume();

  if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
    AppHibernationManager hibernationManager = getSystemService(AppHibernationManager.class);

    if (hibernationManager != null &&
        hibernationManager.isHibernatingForUser(getPackageName())) {
      // App is recovering from hibernation
      Log.d("Hibernation", "App is recovering from hibernation");
```

```
            // Reinitialize necessary components
            reinitializeComponents();

            // Request necessary permissions (might have been revoked during hibernation)
            checkAndRequestPermissions();

            // Rebuild cache
            rebuildCache();
        }
    }
}
```

16. Bluetooth Permission Changes

Change point: Bluetooth permission granularity

Android 12 has subdivided Bluetooth permissions, enhancing user privacy protection.

```
// Request Bluetooth permissions for Android 10 and Android 12
private void requestBluetoothPermissions() {
    List<String> permissionsList = new ArrayList<>();

    // DIFFERENCE: Android 12 requires explicit Bluetooth permissions
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
        // New Bluetooth permissions for Android 12
        permissionsList.add(Manifest.permission.BLUETOOTH_SCAN);
        permissionsList.add(Manifest.permission.BLUETOOTH_CONNECT);
        permissionsList.add(Manifest.permission.BLUETOOTH_ADVERTISE);
    } else {
        // Android 10 and below require location permission for Bluetooth scanning
        permissionsList.add(Manifest.permission.ACCESS_FINE_LOCATION);
    }

    String[] permissions = permissionsList.toArray(new String[0]);
    requestPermissions(permissions, REQUEST_BLUETOOTH_PERMISSIONS);
```

```
}

// Start Bluetooth scanning for Android 10 and Android 12
private void startBluetoothScan() {
   BluetoothAdapter bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
   if (bluetoothAdapter != null && bluetoothAdapter.isEnabled()) {
      // DIFFERENCE: Check for new permissions in Android 12
      if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
         if (checkSelfPermission(Manifest.permission.BLUETOOTH_SCAN) !=
               PackageManager.PERMISSION_GRANTED) {
            // No permission, request permission
            return;
         }
      }

      BluetoothLeScanner scanner = bluetoothAdapter.getBluetoothLeScanner();
      scanner.startScan(scanCallback);
   }
}
```

Main changes:

1. Permission subdivision: Bluetooth functionality is split into BLUETOOTH_SCAN, BLUETOOTH_CONNECT, and BLUETOOTH_ADVERTISE three separate permissions
2. Separation from location permission: Bluetooth scanning no longer automatically requires location permission
3. Precise control: Users can more precisely control app's Bluetooth access scope
4. Runtime permission: All Bluetooth permissions are runtime permissions, need to be requested dynamically
5. Permission description: Need to provide permission usage reason in manifest file

17. Component Export Changes

Change Content: Android 12 requires all components to explicitly declare their exported attribute, no longer allowing implicit export state determination based on intent filters.

Reference Link: https://developer.android.com/about/versions/12/behavior-changes-all#exported

Code Examples:

```
// Android 10: Send broadcast
private void sendBroadcast() {
    Intent intent = new Intent("com.example.app.CUSTOM_ACTION");
    intent.putExtra("data", "some_data");
    sendBroadcast(intent);
}

// Android 12: Send internal and public broadcasts
private void sendInternalBroadcast() {
    Intent intent = new Intent("com.example.app.CUSTOM_ACTION");
    intent.putExtra("data", "some_data");
    // DIFFERENCE: Specify receiver package
    intent.setPackage(getPackageName());
    sendBroadcast(intent);
}

private void sendPublicBroadcast() {
    Intent intent = new Intent("com.example.app.PUBLIC_ACTION");
    intent.putExtra("data", "public_data");
    // DIFFERENCE: Specify required permission
    sendBroadcast(intent, "com.example.app.CUSTOM_PERMISSION");
}

// Access content provider for Android 10 and Android 12
private void accessContentProvider() {
    // Android 12: Check for permission before accessing
    if (ContextCompat.checkSelfPermission(this, "com.example.app.READ_PROVIDER")
```

```
        == PackageManager.PERMISSION_GRANTED) {
        ContentResolver resolver = getContentResolver();
        Uri uri = Uri.parse("content://com.example.app.provider/items");
        Cursor cursor = resolver.query(uri, null, null, null, null);
        // Process query results...
        if (cursor != null) {
            cursor.close();
        }
    } else {
        // No permission, cannot access
        Log.e("ContentProvider", "No permission to access content provider");
    }
}


<!-- AndroidManifest.xml -->
<!-- Android 10: No need to specify exported attribute -->
<receiver android:name=".MyBroadcastReceiver">
    <intent-filter>
        <action android:name="com.example.app.CUSTOM_ACTION" />
    </intent-filter>
</receiver>

<provider
    android:name=".MyContentProvider"
    android:authorities="com.example.app.provider" />

<!-- Android 12: Must specify exported attribute -->
<!-- Set to false for internal components -->
<receiver
    android:name=".MyBroadcastReceiver"
    android:exported="false"> <!-- DIFFERENCE: Specify exported attribute -->
    <intent-filter>
        <action android:name="com.example.app.CUSTOM_ACTION" />
    </intent-filter>
</receiver>
```

```
<!-- Set to true for components accessed by other apps, with permission protection -->
<provider
   android:name=".MyContentProvider"
   android:authorities="com.example.app.provider"
   android:exported="true" <!-- DIFFERENCE: Specify exported attribute -->
   android:permission="com.example.app.READ_PROVIDER"
   android:writePermission="com.example.app.WRITE_PROVIDER" />

<!-- Declare custom permissions for exported components -->
<permission
   android:name="com.example.app.READ_PROVIDER"
   android:protectionLevel="signature" />
<permission
   android:name="com.example.app.WRITE_PROVIDER"
   android:protectionLevel="signature" />
```

18. Microphone and Camera Toggle

Change Content: Android 12 adds global toggles for microphone and camera in Quick Settings, requiring apps to handle these toggle states appropriately.

Reference Link: https://developer.android.com/about/versions/12/behavior-changes-12#mic-camera-toggles

Code Examples:

```
// Capture photo for Android 10 and Android 12
private void capturePhoto() {
   if (ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA)
         == PackageManager.PERMISSION_GRANTED) {

      CameraManager cameraManager = (CameraManager) getSystemService(Context.CAMERA_SERVICE);

      // DIFFERENCE: Check system-level camera toggle in Android 12
```

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
    try {
        String cameraId = cameraManager.getCameraIdList()[0];
        cameraManager.openCamera(cameraId, new CameraDevice.StateCallback() {
            @Override
            public void onOpened(@NonNull CameraDevice camera) {
                // Camera opened successfully
                mCameraDevice = camera;
                createCameraPreviewSession();
            }

            @Override
            public void onDisconnected(@NonNull CameraDevice camera) {
                camera.close();
                mCameraDevice = null;
            }

            @Override
            public void onError(@NonNull CameraDevice camera, int error) {
                camera.close();
                mCameraDevice = null;

                if (error == CameraDevice.StateCallback.ERROR_CAMERA_DISABLED) {
                    // Camera is disabled by system
                    Toast.makeText(MainActivity.this,
                            "Camera is disabled by system, please enable in settings",
                            Toast.LENGTH_LONG).show();
                    showCameraDisabledDialog();
                } else {
                    Toast.makeText(MainActivity.this,
                            "Camera error: " + error,
                            Toast.LENGTH_SHORT).show();
                }
            }
        }, null);
    } catch (CameraAccessException | SecurityException e) {
```

```
            Log.e("Camera", "Cannot access camera", e);
            Toast.makeText(this, "Cannot access camera, please check system settings",
                    Toast.LENGTH_SHORT).show();
            showCameraDisabledDialog();
        }
    } else {
        // Camera usage for Android 11 and below
        try {
            String cameraId = cameraManager.getCameraIdList()[0];
            cameraManager.openCamera(cameraId, cameraStateCallback, null);
        } catch (CameraAccessException e) {
            Log.e("Camera", "Cannot access camera", e);
        }
    }
} else {
    // Request camera permission
    ActivityCompat.requestPermissions(this,
            new String[]{Manifest.permission.CAMERA},
            REQUEST_CAMERA_PERMISSION);
}
}

// Show camera disabled dialog
private void showCameraDisabledDialog() {
    new AlertDialog.Builder(this)
        .setTitle("Camera Disabled")
        .setMessage("Camera is disabled by system. Please go to Settings > Privacy > Camera toggle to enable
camera access.")
        .setPositiveButton("Go to Settings", (dialog, which) -> {
            Intent intent = new Intent(Settings.ACTION_PRIVACY_SETTINGS);
            startActivity(intent);
        })
        .setNegativeButton("Cancel", null)
        .show();
}
```

```
// Start recording for Android 10 and Android 12
private void startRecording() {
   if (ContextCompat.checkSelfPermission(this, Manifest.permission.RECORD_AUDIO)
         == PackageManager.PERMISSION_GRANTED) {

      audioRecorder = new MediaRecorder();

      try {
         audioRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
         audioRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
         audioRecorder.setOutputFile(getRecordingFilePath());
         audioRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);

         audioRecorder.prepare();

         // DIFFERENCE: Check system-level microphone toggle in Android 12
         try {
            audioRecorder.start();
            isRecording = true;

            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
               Handler handler = new Handler();
               handler.postDelayed(() -> {
                  if (audioRecorder != null) {
                     try {
                        int amplitude = audioRecorder.getMaxAmplitude();
                        if (amplitude <= 0 && isRecording) {
                           Log.w("AudioRecorder", "Recording amplitude is 0, microphone might be disabled");
                        }
                     } catch (Exception e) {
                        // Ignore
                     }
                  }
               }, 1000);
            }
         } catch (IllegalStateException e) {
```

```
            Log.e("AudioRecorder", "Failed to start recording", e);
            Toast.makeText(this, "Cannot start recording, please check if microphone is disabled",
                    Toast.LENGTH_SHORT).show();
            releaseMediaRecorder();
            showMicrophoneDisabledDialog();
        }
    } catch (IOException | IllegalArgumentException | IllegalStateException e) {
        Log.e("AudioRecorder", "Recording setup failed", e);
        releaseMediaRecorder();
        if (e instanceof IllegalArgumentException || e instanceof IllegalStateException) {
            Toast.makeText(this, "Cannot use microphone, please check system settings",
                    Toast.LENGTH_SHORT).show();
            showMicrophoneDisabledDialog();
        }
    }
  } else {
    // Request recording permission
    ActivityCompat.requestPermissions(this,
            new String[]{Manifest.permission.RECORD_AUDIO},
            REQUEST_RECORD_AUDIO_PERMISSION);
  }
}

// Show microphone disabled dialog
private void showMicrophoneDisabledDialog() {
  new AlertDialog.Builder(this)
        .setTitle("Microphone Disabled")
        .setMessage("Microphone is disabled by system. Please go to Settings > Privacy > Microphone toggle to
enable microphone access.")
        .setPositiveButton("Go to Settings", (dialog, which) -> {
            Intent intent = new Intent(Settings.ACTION_PRIVACY_SETTINGS);
            startActivity(intent);
        })
        .setNegativeButton("Cancel", null)
        .show();
}
```

```
// Release MediaRecorder resources
private void releaseMediaRecorder() {
    if (audioRecorder != null) {
        if (isRecording) {
            try {
                audioRecorder.stop();
            } catch (Exception e) {
                // Ignore stop exceptions
            }
            isRecording = false;
        }
        audioRecorder.reset();
        audioRecorder.release();
        audioRecorder = null;
    }
}
```

19. Clipboard Access Toast

Change Content: Android 12 shows a notification when an app accesses clipboard content, requiring apps to be explicit about when they need to access the clipboard.

Reference Link: https://developer.android.com/about/versions/12/behavior-changes-12#clipboard-access

Code Examples:

```
// Android 10: Access clipboard anytime without notification
private void pasteFromClipboard() {
    ClipboardManager clipboard = (ClipboardManager) getSystemService(Context.CLIPBOARD_SERVICE);

    // Read clipboard without user interaction
    if (clipboard.hasPrimaryClip()) {
        ClipData clipData = clipboard.getPrimaryClip();
```

```
        if (clipData != null && clipData.getItemCount() > 0) {
            CharSequence text = clipData.getItemAt(0).getText();
            if (text != null) {
                // Auto-fill text to input field
                EditText editText = findViewById(R.id.edit_text);
                editText.setText(text);
            }
        }
    }
}

// Android 12: Access clipboard only when explicitly requested by user
private void setupPasteButton() {
    Button pasteButton = findViewById(R.id.paste_button);
    EditText editText = findViewById(R.id.edit_text);

    // Provide explicit paste button
    pasteButton.setOnClickListener(v -> {
        // Access clipboard when user explicitly clicks paste button
        ClipboardManager clipboard = (ClipboardManager) getSystemService(Context.CLIPBOARD_SERVICE);

        if (clipboard.hasPrimaryClip()) {
            ClipData clipData = clipboard.getPrimaryClip();
            if (clipData != null && clipData.getItemCount() > 0) {
                CharSequence text = clipData.getItemAt(0).getText();
                if (text != null) {
                    // Fill text to input field
                    editText.setText(text);
                }
            }
        }
    });

    // Use system paste menu
    editText.setOnLongClickListener(v -> {
        // Show context menu on long press, including system paste option
```

```
        return false; // Allow system to handle long press
    });
}

// Android 10: Automatically read clipboard on app launch
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    // Read clipboard on app launch
    pasteFromClipboard();

    // Monitor clipboard changes
    ClipboardManager clipboard = (ClipboardManager) getSystemService(Context.CLIPBOARD_SERVICE);
    clipboard.addPrimaryClipChangedListener(() -> {
        // Automatically read clipboard when content changes
        pasteFromClipboard();
    });
}

// Android 12: Avoid automatically monitoring clipboard changes
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    // Set up paste button
    setupPasteButton();

    // Don't read clipboard automatically on app launch
    // Don't register clipboard change listener to automatically read content
}

// Check for relevant content when specific conditions are met
private void checkForRelevantContent() {
```

```
    // Only access clipboard in specific scenarios
    EditText trackingField = findViewById(R.id.tracking_number);
    if (trackingField.hasFocus()) {
        ClipboardManager clipboard = (ClipboardManager) getSystemService(Context.CLIPBOARD_SERVICE);

        if (clipboard.hasPrimaryClip()) {
            ClipData clipData = clipboard.getPrimaryClip();
            if (clipData != null && clipData.getItemCount() > 0) {
                CharSequence text = clipData.getItemAt(0).getText();
                if (text != null && isTrackingNumberFormat(text.toString())) {
                    // Notify user they can paste this content
                    Toast.makeText(this, "Detected tracking number in clipboard, tap paste button to use",
                            Toast.LENGTH_SHORT).show();
                    findViewById(R.id.paste_button).setVisibility(View.VISIBLE);
                }
            }
        }
    }
}

// Check if text matches tracking number format
private boolean isTrackingNumberFormat(String text) {
    // Implement tracking number format validation logic
    return text.matches("\\d{12,15}") ||
           text.matches("[A-Z]{2}\\d{9}[A-Z]{2}");
}
```

20. Safer Component Exporting

Change Content: Android 12 requires apps to export components more securely, including providing stricter permissions and intent filters.

Reference Link: https://developer.android.com/about/versions/12/behavior-changes-all#exported

Code Examples:

```
// Android 10: Send broadcast
private void sendBroadcast() {
    Intent intent = new Intent("com.example.app.CUSTOM_ACTION");
    intent.putExtra("data", "some_data");
    sendBroadcast(intent);
}

// Android 12: Send internal and public broadcasts
private void sendInternalBroadcast() {
    Intent intent = new Intent("com.example.app.CUSTOM_ACTION");
    intent.putExtra("data", "some_data");
    // DIFFERENCE: Specify receiver package
    intent.setPackage(getPackageName());
    sendBroadcast(intent);
}

private void sendPublicBroadcast() {
    Intent intent = new Intent("com.example.app.PUBLIC_ACTION");
    intent.putExtra("data", "public_data");
    // DIFFERENCE: Specify required permission
    sendBroadcast(intent, "com.example.app.CUSTOM_PERMISSION");
}

// Access content provider for Android 10 and Android 12
private void accessContentProvider() {
    // Android 12: Check for permission before accessing
    if (ContextCompat.checkSelfPermission(this, "com.example.app.READ_PROVIDER")
            == PackageManager.PERMISSION_GRANTED) {
        ContentResolver resolver = getContentResolver();
        Uri uri = Uri.parse("content://com.example.app.provider/items");
        Cursor cursor = resolver.query(uri, null, null, null, null);
        // Process query results...
        if (cursor != null) {
            cursor.close();
        }
    } else {
```

```
        // No permission, cannot access
        Log.e("ContentProvider", "No permission to access content provider");
    }
}


<!-- AndroidManifest.xml -->
<!-- Android 10: Declare broadcast receiver without specifying exported attribute -->
<receiver android:name=".MyBroadcastReceiver">
    <intent-filter>
        <action android:name="com.example.app.CUSTOM_ACTION" />
    </intent-filter>
</receiver>

<provider
    android:name=".MyContentProvider"
    android:authorities="com.example.app.provider" />

<!-- Android 12: Must specify exported attribute -->
<!-- Internal broadcast receiver -->
<receiver
    android:name=".MyBroadcastReceiver"
    android:exported="false"> <!-- DIFFERENCE: Specify exported attribute -->
    <intent-filter>
        <action android:name="com.example.app.CUSTOM_ACTION" />
    </intent-filter>
</receiver>

<!-- Public broadcast receiver with permission -->
<receiver
    android:name=".PublicBroadcastReceiver"
    android:exported="true" <!-- DIFFERENCE: Specify exported attribute -->
    android:permission="com.example.app.CUSTOM_PERMISSION">
    <intent-filter>
        <action android:name="com.example.app.PUBLIC_ACTION" />
    </intent-filter>
```

```
</receiver>

<!-- Content provider with permissions -->
<provider
    android:name=".MyContentProvider"
    android:authorities="com.example.app.provider"
    android:exported="true" <!-- DIFFERENCE: Specify exported attribute -->
    android:permission="com.example.app.READ_PROVIDER"
    android:writePermission="com.example.app.WRITE_PROVIDER" />

<!-- Declare custom permissions -->
<permission
    android:name="com.example.app.CUSTOM_PERMISSION"
    android:protectionLevel="signature" />
<permission
    android:name="com.example.app.READ_PROVIDER"
    android:protectionLevel="signature" />
<permission
    android:name="com.example.app.WRITE_PROVIDER"
    android:protectionLevel="signature" />
```

21. Material Design Component Updates

Change Point: Material You Design Language

Android 12 introduces Material You design language, providing a more personalized UI experience, including a dynamic color system.

```
<!-- Android 10: Use fixed theme colors -->
<style name="Theme.App" parent="Theme.MaterialComponents">
    <item name="colorPrimary">@color/primary</item>
    <item name="colorPrimaryDark">@color/primary_dark</item>
    <item name="colorAccent">@color/accent</item>
</style>
```

```
<!-- Android 12: Use dynamic colors in themes.xml -->
<style name="Theme.App" parent="Theme.Material3.DayNight">
    <item name="colorPrimary">@color/material_dynamic_primary40</item>
    <item name="colorPrimaryDark">@color/material_dynamic_primary80</item>
    <item name="colorAccent">@color/material_dynamic_secondary40</item>
</style>


// Android 10: Use fixed colors in code
button.setBackgroundColor(getResources().getColor(R.color.primary));

// Android 12: Get and use dynamic colors in code
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
    // DIFFERENCE: Get system-extracted dynamic colors
    Context context = getContext();
    TypedArray dynamicColors = context.obtainStyledAttributes(
            new int[] {
                android.R.attr.colorPrimary,
                android.R.attr.colorAccent
            });
    int dynamicPrimary = dynamicColors.getColor(0, 0);
    int dynamicAccent = dynamicColors.getColor(1, 0);
    dynamicColors.recycle();

    // Apply dynamic colors
    button.setBackgroundColor(dynamicPrimary);
}
```

Main changes:

1. Dynamic color system: Automatically extracts color scheme from user's wallpaper
2. Material You components: Updated UI components supporting more rounded shapes and dynamic themes
3. Personalized themes: Allows apps to adapt to user's personalization preferences

4. Material 3: New design system providing a more modern visual language

5. Adaptive layouts: Better adaptation to different screen sizes and shapes

22. Widget Improvements

Change Point: Widget API Updates

Android 12 has made major updates to the widget system, providing a more modern appearance and better user experience.

```
// Widget implementation for Android 10 and Android 12
public class ExampleAppWidget extends AppWidgetProvider {
    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {
        for (int appWidgetId : appWidgetIds) {
            RemoteViews views = new RemoteViews(context.getPackageName(), R.layout.example_widget);
            views.setTextViewText(R.id.widget_text, "Widget Example");

            // Set click event
            Intent intent = new Intent(context, MainActivity.class);
            // DIFFERENCE: Use PendingIntent.FLAG_IMMUTABLE for Android 12
            PendingIntent pendingIntent = PendingIntent.getActivity(
                context, 0, intent, PendingIntent.FLAG_IMMUTABLE);
            views.setOnClickPendingIntent(R.id.widget_layout, pendingIntent);

            // DIFFERENCE: Use new rounded corners API for Android 12
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
                views.setViewOutlinePreference(R.id.widget_layout,
                    RemoteViews.OUTLINE_PREFERENCE_ROUNDED);
            }

            appWidgetManager.updateAppWidget(appWidgetId, views);
        }
    }
```

```
}
```

Android 12 Implementation:

```
<!-- widget_info.xml for Android 10 and Android 12 -->
<appwidget-provider
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="40dp"
    android:minHeight="40dp"
    android:updatePeriodMillis="1800000"
    android:initialLayout="@layout/example_widget"
    android:resizeMode="horizontal|vertical"
    android:widgetCategory="home_screen"
    android:description="@string/widget_description" <!-- DIFFERENCE: Add description for Android 12 -->
    android:targetCellWidth="2" <!-- DIFFERENCE: Specify target cell dimensions for Android 12 -->
    android:targetCellHeight="1">
</appwidget-provider>
```

Main changes:

1. Rounded corners support: Widgets now support rounded corners, consistent with system UI style
2. Responsive layout: Improved layout system for better size adaptation
3. Dynamic colors: Support for Material You dynamic color system
4. Performance improvements: Optimized update mechanism for smoother experience
5. New attributes: Added targetCellWidth and targetCellHeight attributes for better widget size control
6. PendingIntent security: Must use FLAG_IMMUTABLE or FLAG_MUTABLE flags

23. Screenshot and Screen Recording

Change Point: Screenshot and Screen Recording APIs

Android 12 introduces new screenshot and screen recording APIs, while also adding privacy protection measures.

```
// Android 10: Screen capture using MediaProjection
private void startScreenCapture() {
    MediaProjectionManager projectionManager =
        (MediaProjectionManager) getSystemService(Context.MEDIA_PROJECTION_SERVICE);
    startActivityForResult(projectionManager.createScreenCaptureIntent(), REQUEST_CODE);
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == REQUEST_CODE && resultCode == RESULT_OK) {
        // Use MediaProjection for screen recording
        MediaProjection mediaProjection = projectionManager.getMediaProjection(resultCode, data);
        // Set recording parameters...
    }
}

// Android 12: Screenshot API
@RequiresApi(api = Build.VERSION_CODES.S)
private void takeScreenshot() {
    // Create screenshot callback
    ScreenshotCallback screenshotCallback = new ScreenshotCallback() {
        @Override
        public void onScreenCaptured(Bitmap bitmap, Rect source) {
            // Handle screenshot
            // bitmap contains screenshot content
        }

        @Override
        public void onScreenCaptureError(int errorCode) {
            // Handle error
        }
    };
```

```
    // Request screenshot
    ScreenshotManager screenshotManager =
        getSystemService(ScreenshotManager.class);
    screenshotManager.takeScreenshot(
        WindowManager.ScreenshotSource.SCREENSHOT_OTHER,
        new Handler(Looper.getMainLooper()),
        screenshotCallback);
}

// Android 12: Screen capture with privacy indicator
@RequiresApi(api = Build.VERSION_CODES.S)
private void startScreenCapture() {
    MediaProjectionManager projectionManager =
        (MediaProjectionManager) getSystemService(Context.MEDIA_PROJECTION_SERVICE);
    startActivityForResult(projectionManager.createScreenCaptureIntent(), REQUEST_CODE);
    // DIFFERENCE: Android 12 shows a recording indicator in the status bar
}
```

Main changes:

1. Official Screenshot API: Provides native screen capture API
2. Privacy Indicator: Shows permanent indicator when recording screen
3. Background Access Restriction: Limits background apps' access to screen content
4. Security Enhancement: Added protection mechanisms for sensitive content
5. Screenshot Editing: System screenshot tool provides more editing options

24. Widget API Updates

Change Content: Android 12 has made major updates to the widget system, providing a more modern appearance and better user experience, including rounded corners support, responsive layouts, and dynamic color support.

Reference Link: https://developer.android.com/about/versions/12/features#widgets

Code Examples:

```
// Widget implementation for Android 10 and Android 12
public class ExampleAppWidget extends AppWidgetProvider {
    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {
        for (int appWidgetId : appWidgetIds) {
            RemoteViews views = new RemoteViews(context.getPackageName(), R.layout.example_widget);
            views.setTextViewText(R.id.widget_text, "Widget Example");

            // Set click event
            Intent intent = new Intent(context, MainActivity.class);
            // DIFFERENCE: Use PendingIntent.FLAG_IMMUTABLE for Android 12
            PendingIntent pendingIntent = PendingIntent.getActivity(
                context, 0, intent, PendingIntent.FLAG_IMMUTABLE);
            views.setOnClickPendingIntent(R.id.widget_layout, pendingIntent);

            // DIFFERENCE: Use new rounded corners API for Android 12
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
                views.setViewOutlinePreference(R.id.widget_layout,
                    RemoteViews.OUTLINE_PREFERENCE_ROUNDED);
            }

            appWidgetManager.updateAppWidget(appWidgetId, views);
        }
    }
}


<!-- widget_info.xml for Android 10 and Android 12 -->
<appwidget-provider
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="40dp"
    android:minHeight="40dp"
```

```
    android:updatePeriodMillis="1800000"
    android:initialLayout="@layout/example_widget"
    android:resizeMode="horizontal|vertical"
    android:widgetCategory="home_screen"
    android:description="@string/widget_description" <!-- DIFFERENCE: Add description for Android 12 -->
    android:targetCellWidth="2" <!-- DIFFERENCE: Specify target cell dimensions for Android 12 -->
    android:targetCellHeight="1">
</appwidget-provider>
```

25. Notification Template and Style Updates

Change Content: Android 12 has updated the notification system visually and functionally to align with Material You design language, including larger touch targets, adaptive colors, and improved media controls.

Reference Link: https://developer.android.com/about/versions/12/features#notifications

Code Examples:

```
// Notification implementation for Android 10 and Android 12
private void showNotification() {
    NotificationCompat.Builder builder = new NotificationCompat.Builder(context, CHANNEL_ID)
        .setSmallIcon(R.drawable.notification_icon)
        .setContentTitle("Notification Title")
        .setContentText("Notification Content")
        .setPriority(NotificationCompat.PRIORITY_DEFAULT);

    // DIFFERENCE: Use richer styles in Android 12
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
        builder.setStyle(new NotificationCompat.DecoratedCustomViewStyle());
        // On Android 12, notification visual style will automatically adapt to Material You design
    }

    NotificationManagerCompat notificationManager = NotificationManagerCompat.from(context);
    notificationManager.notify(notificationId, builder.build());
```

```
}
```

26. Rendering Pipeline Changes

Change Content: Android 12 changes the rendering pipeline, which may affect certain custom UI implementations. It introduces new frame rate control APIs and provides different compatibility modes.

Reference Link: https://developer.android.com/about/versions/12/behavior-changes-all?hl=en#rendering-pipeline

Code Examples:

```
// Custom view implementation for Android 10 and Android 12
public class CustomView extends View {
    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        // Drawing operations
        canvas.drawRect(...);

        // DIFFERENCE: Be aware of different rendering behaviors in Android 12
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
            // Consider rendering optimizations or changes in behavior
        }
    }

    // Android 10: Use Choreographer for frame rate control
    private void setFrameRate() {
        Choreographer.getInstance().postFrameCallback(new Choreographer.FrameCallback() {
            @Override
            public void doFrame(long frameTimeNanos) {
                // Handle frame update
                invalidate();
                // Continue requesting next frame
```

```
            Choreographer.getInstance().postFrameCallback(this);
        }
    });
}

// Android 12: Use Surface.setFrameRate for controlling render frame rate
@RequiresApi(api = Build.VERSION_CODES.S)
private void setFrameRate(Surface surface) {
    // Set fixed frame rate mode
    surface.setFrameRate(60.0f,
            Surface.FRAME_RATE_COMPATIBILITY_FIXED_SOURCE,
            Surface.CHANGE_FRAME_RATE_ALWAYS);
}
}
```