Report of Where Am I Application

Tamer KARALÜRT

155050010

Atacan GÜLER

155050041

DECEMBER 2017

İçindekiler

1)What is Where am I and what does it do?

Where am I is a location keeper in your phone. It saves the locations that you've been to a database as soon as you log in to the app.

User needs an account to use the app. App logs the movement. Where am I stores the location for every 3 seconds. Data is stored on a database and in a txt file as well. The stored locations queued with Listview. User can see where he/she was with this tool. Any location that is stored on listview can show the exact location on Google maps with just a click so user can see where he/she was.

2)The Software Development Tools and Tools Used to Make Where am I: Android Studio 3.0 and Features

The new Android Profiler window in Android Studio 3.0 replaces the Android Monitor tools. These new profiling tools provide realtime data for your app's CPU, memory, and network activity. You can perform sample-based method tracing to time your code execution, capture heap dumps, view memory allocations, and inspect the details of network-transmitted files.

Android Profiler now displays a shared timeline view, which includes a timeline with realtime graphs for CPU, memory, and network usage. The window also includes timeline zoom controls , a button to jump forward to the realtime updates , and an event timeline that shows activity states, user input events, and screen rotation events .

3)Java and Features

There is given many features of java. They are also known as java buzzwords. The Java Features given below are simple and easy to understand.

1. Simple
2. Object-Oriented
3. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral

8. Dynamic
9. Interpreted
10. High Performance
11. Multithreaded

3.1) Simple

According to Sun, Java language is simple because:

syntax is based on C++ (so easier for programmers to learn it after C++).

removed many confusing and/or rarely-used features e.g., explicit pointers, operator overloading etc.

No need to remove unreferenced objects because there is Automatic Garbage Collection in java.

3.2) Object-oriented

Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behaviour.

Object-oriented programming(OOPs) is a methodology that simplify software development and maintenance by providing some rules.

Basic concepts of OOPs are:

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

3.3) Portable

We may carry the java bytecode to any platform.

3.4) Platform Independent

A platform is the hardware or software environment in which a program runs.

There are two types of platforms software-based and hardware-based. Java provides software-based platform.

The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms. It has two components:

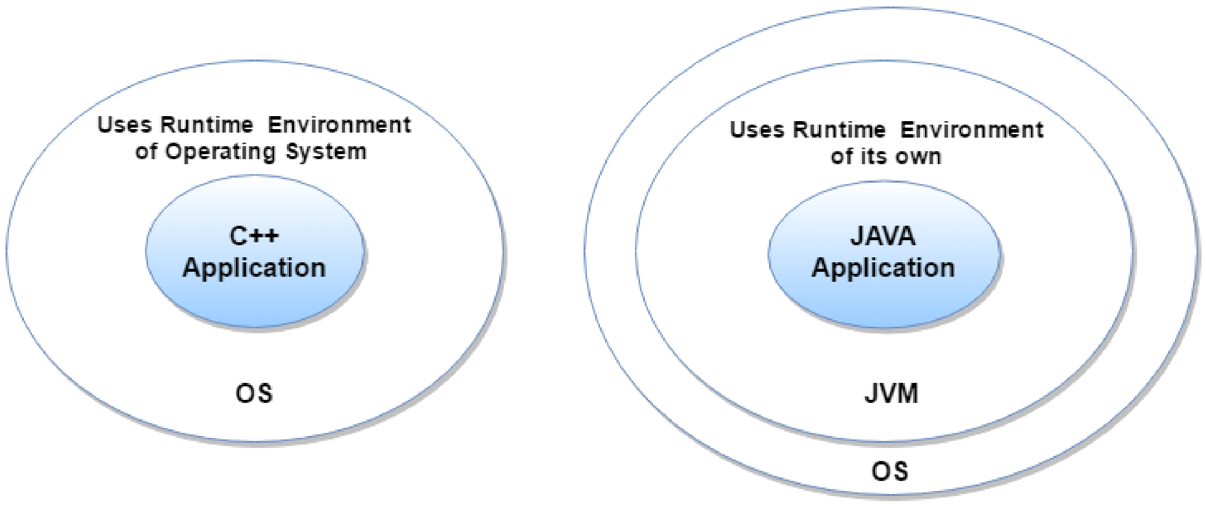
1. Runtime Environment
2. API(Application Programming Interface)

Java code can be run on multiple platforms e.g. Windows, Linux, Sun Solaris, Mac/OS etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere(WORA).

3.5) Secured

Java is secured because:

- No explicit pointer
- Java Programs run inside virtual machine sandbox


Uses Runtime Environment of Operating System
C++ Application
OS
Uses Runtime Environment of its own
JAVA Application
JVM
OS

Classloader: adds security by separating the package for the classes of the local file system from those that are imported from network sources.

Bytecode Verifier: checks the code fragments for illegal code that can violate access right to objects.

Security Manager: determines what resources a class can access such as reading and writing to the local disk.

These security are provided by java language. Some security can also be provided by application developer through SSL, JAAS, Cryptography etc.

3.6) Robust

Robust simply means strong. Java uses strong memory management. There are lack of pointers that avoids security problem. There is automatic garbage collection in java. There is exception handling and type checking mechanism in java. All these points makes java robust.

3.7) Architecture-Neutral

There is no implementation dependent features e.g. size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. But in java, it occupies 4 bytes of memory for both 32 and 64 bit architectures.

3.8) High-Performance

Java is faster than traditional interpretation since byte code is "close" to native code still somewhat slower than a compiled language (e.g., C++)

3.9) Distributed

We can create distributed applications in java. RMI and EJB are used for creating distributed applications. We may access files by calling the methods from any machine on the internet.

3.10) Multi-Threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications etc.

4) What is Latitude and Longitude?

Latitude and Longitude are the units that represent the coordinates at geographic coordinate system. Just like every actual house has its address (which includes the number, the name of the street, city, etc), every single point on the surface of earth can be specified by the latitude and longitude coordinates. Therefore, by using latitude and longitude we can specify virtually any point on earth.

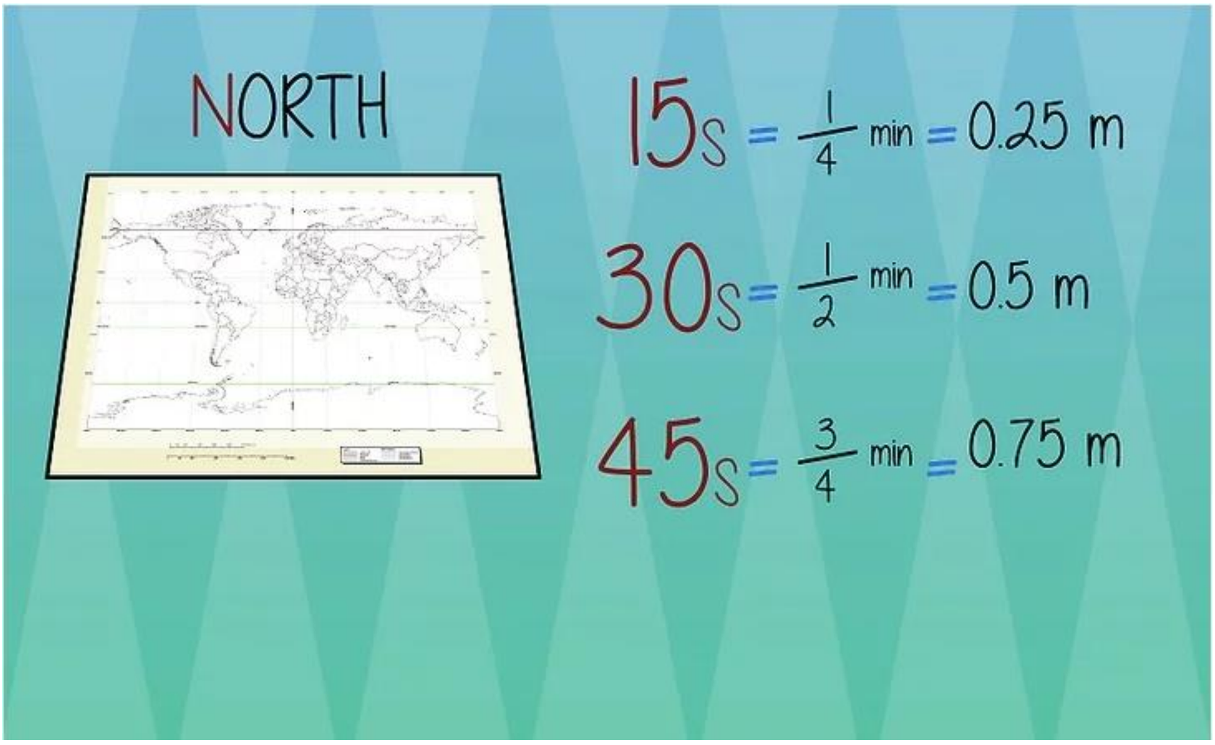
The latitude has the symbol of phi, and it shows the angle between the straight line in the certain point and the equatorial plane. The latitude is specified by degrees, starting from 0° and ending up with 90° to both sides of the equator, making latitude Northern and Southern. The equator is the line with 0° latitude. The longitude has the symbol of lambda and is another angular coordinate defining the position of a point on a surface of earth. The longitude is defined as an angle pointing west or east from the Greenwich Meridian, which is taken as the Prime Meridian. The longitude can be defined maximum as 180° east from the Prime Meridian and 180° west from the Prime Meridian.

Both latitude and longitude are measured in degrees, which are in turn divided into minutes and seconds. For example, the tropical zone which is located to the south and to the north from the Equator is determined by the limits of 23°26'13.7" S and 23°26'13.7" N. Or. For example, the geographical coordinates of the mount Ngauruhoe in New Zealand, famous with its being the filming area for the Lord of the Rings movie, has the geographic coordinates of 39°09'24.6"S 175°37'55.8"E.

5) About Lat Long

Latlong.net is an online geographic tool that can be used to lookup latitude and longitude of a place, and get its coordinates on map. You can search for a place using a city's or town's name, as well as the name of special places, and the correct *lat long coordinates* will be shown at the bottom of the latitude longitude finder form. At that, the place you found will be displayed with the point marker centered on map. Also the gps coordinates will be displayed below the map.

Degrees are a large unit of measurement so to get an accurate location, longitude and latitude are divided to the decimal point referred to as decimal degrees. For example, you may see latitude as 35.789 degrees North. Global Position Systems (GPS) often show decimal degrees but printed maps do not.
Online topographic maps express longitude and latitude decimal degrees in degrees, minutes and seconds. Each degree equals 60 minutes, while each minute equals 60 seconds. The comparison to time allows for easier subdivision.



First, always assume that the top of the map is north.The numbers on the right and left side of a map refer to the latitude. The numbers on the top and bottom of the map are the longitude.

How to convert time in order understand a map using decimal degrees as degrees, minutes, and seconds

15 seconds = one quarter of a minute = 0.25 minutes

30 seconds = half a minute = 0.5 minutes

45 seconds = three quarters of a minute = 0.75 minutes

6) SQLite and Features

SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. SQLite is the most widely deployed database in the world with more applications than we can count, including several high-profile projects.

Features:

1. Transactions are atomic, consistent, isolated, and durable (ACID) even after system crashes and power failures.
2. Zero-configuration - no setup or administration needed.
3. Full-featured SQL implementation with advanced capabilities like partial indexes, indexes on expressions, JSON, and common table expressions. (Omitted features)
4. A complete database is stored in a single cross-platform disk file. Great for use as an application file format.
5. Supports terabyte-sized databases and gigabyte-sized strings and blobs.
6. Small code footprint: less than 500KiB fully configured or much less with optional features omitted.
7. Simple, easy to use API.
8. Fast: In some cases, SQLite is faster than direct filesystem I/O
9. Written in ANSI-C. TCL bindings included. Bindings for dozens of other languages available separately.
10. Well-commented source code with 100% branch test coverage.
11. Available as a single ANSI-C source-code file that is easy to compile and hence is easy to add into a larger project.
12. Self-contained: no external dependencies.
13. Cross-platform: Android, *BSD, iOS, Linux, Mac, Solaris, VxWorks, and Windows (Win32, WinCE, WinRT) are supported out of the box. Easy to port to other systems.
14. Sources are in the public domain. Use for any purpose.
15. Comes with a standalone command-line interface (CLI) client that can be used to administer SQLite databases

7) Android 4.0.3 APIs

Android 4.0.3 (ICE_CREAM_SANDWICH_MR1) is an incremental release of the Android 4.0 (ICE_CREAM_SANDWICH) platform family. This release includes new features for users and developers, API changes, and various bug fixes.

For developers, the Android 4.0.3 platform is available as a downloadable component for the Android SDK. The downloadable platform includes an Android library and system image, as well as a set of emulator skins and more. To get started developing or testing against Android 4.0.3, use the Android SDK Manager to download the platform into your SDK.

7.1) API Overview

The sections below provide a technical overview of new APIs in Android 4.0.3.

7.2) Social Stream API in Contacts Provider

Applications that use social stream data such as status updates and check-ins can now sync that data with each of the user's contacts, providing items in a stream along with photos for each.

The database table that contains an individual contact's social stream is defined by android.provider.ContactsContract.StreamItems, the Uri for which is nested within the ContactsContract.RawContacts directory to which the stream items belong. Each social stream table includes several columns for metadata about each stream item, such as an icon representing the source (an avatar), a label for the item, the primary text content, comments about the item (such as responses from other people), and more. Photos associated with a stream are stored in another table, defined by android.provider.ContactsContract.StreamItemPhotos, which is available as a sub-directory of the android.provider.ContactsContract.StreamItems Uri.

To read or write social stream items for a contact, an application must request permission from the user by declaring <uses-permission android:name="android.permission.READ_SOCIAL_STREAM"> and/or <uses-permission android:name="android.permission.WRITE_SOCIAL_STREAM">in their manifest files.

7.3) Calendar Provider

- Adds the class CalendarContract.Colors to represent a color table in the Calendar Provider. The class provides fields for accessing colors available for a given account. Colors are referenced by COLOR_KEY which must be unique for a given account name/type. These values can only be updated by the sync adapter.
- Adds ALLOWED_AVAILABILITY and ALLOWED_ATTENDEE_TYPES for exchange/sync support.
- Adds TYPE_RESOURCE (such as conference rooms) for attendees and AVAILABILITY_TENTATIVE, as well as EVENT_COLOR_KEY for events.

8) Home Screen Widgets

Starting from Android 4.0, home screen widgets should no longer include their own padding. Instead, the system now automatically adds padding for each widget, based the characteristics of the current screen. This leads to a more uniform, consistent presentation of widgets in a grid. To assist applications that host home screen widgets, the platform provides a new method getDefaultPaddingForWidget(). Applications can call this method to get the system-defined padding and account for it when computing the number of cells to allocate to the widget.

9) Spell-Checking

- For apps that accessing spell-checker services, a new cancel() method cancels any pending and running spell-checker tasks in a session.
- For spell-checker services, a new suggestions flag, RESULT_ATTR_HAS_RECOMMENDED_SUGGESTIONS, lets the services distinguish higher-confidence suggestions from lower-confidence ones. For example, a spell-checker could set the flag if an input word is not in the user dictionary but has likely suggestions, or not set the flag if an input word is not in the dictionary and has suggestions that are likely to be less useful.

  Apps connected to the spell-checker can use the RESULT_ATTR_HAS_RECOMMENDED_SUGGESTIONS flag in combination with other suggestion attributes, as well as the getSuggestionsAttributes() and getSuggestionsCount() methods, to determine whether to mark input words as typos and offer suggestions.

- A new FLAG_AUTO_CORRECTION style for text spans indicates that auto correction is about to be applied to a word/text that the user is typing/composing. This type of suggestion is rendered differently, to indicate the auto correction is happening.

10) Bluetooth

New public methods fetchUuidsWithSdp() and getUuids() let apps determine the features (UUIDs) supported by a remote device. In the case of fetchUuidsWithSdp(), the system performs a service discovery on the remote device to get the UUIDs supported, then broadcasts the result in an ACTION_UUID intent.

11) UI Toolkit

New methods setUserVisibleHint() and getUserVisibleHint() allow a fragment to set a hint of whether or not it is currently user-visible. The system defers the start of fragments that are not user-visible until the loaders for visible fragments have run. The visibility hint is "true" by default.

12) Graphics

- New method setDefaultBufferSize(int, int) in SurfaceTexture sets the default size of the image buffers. This method may be used to set the image size when producing images with Canvas (via lockCanvas(Rect)), or OpenGL ES (via an EGLSurface).
- Adds definitions for the enums of the GL_OES_EGL_image_external OpenGL ES extension — GL_REQUIRED_TEXTURE_IMAGE_UNITS_OES,GL_SAMPLER_EXTERNAL_OES, GL_TEXTURE_BINDING_EXTERNAL_OES, and GL_TEXTURE_EXTERNAL_OES.

13) Accessibility

- Clients of RemoteViews can now use the method setContentDescription() to set and get the content description of any View in the inflated layout.
- The methods getMaxScrollX(), getMaxScrollY(), setMaxScrollX(), and setMaxScrollY() allow apps to get and set the maximum scroll offset for an AccessibilityRecord object.
- When touch-exploration mode is enabled, a new secure setting ACCESSIBILITY_SPEAK_PASSWORD indicates whether the user requests the

IME to speak text entered in password fields, even when a headset is not in use. By default, no password text is spoken unless a headset is in use.

14) Text-to-speech

- Adds the new method getFeatures()for querying and enabling network TTS support.
- Adds a new listener class, UtteranceProgressListener, that engines can register to receive notification of speech-synthesis errors.

15) Database

- A new CrossProcessCursorWrapper class lets content providers return results for a cross-process query more efficiently. The new class is a useful building block for wrapping cursors that will be sent to processes remotely. It can also transform normal Cursor objects into CrossProcessCursorobjects transparently.

  The CrossProcessCursorWrapper class fixes common performance issues and bugs that applications have encountered when implementing content providers.
- The CursorWindow(java.lang.String) constructor now takes a name string as input. The system no longer distinguishes between local and remote cursor windows, so CursorWindow(boolean) is now deprecated.

16) Intents

Adds new categories for targeting common types of applications on the device, such as CATEGORY_APP_BROWSER, CATEGORY_APP_CALENDAR, CATEGORY_APP_ MAPS, and more.

17) Camera

- MediaMetadataRetriever adds the new constant METADATA_KEY_LOCATION to let apps access retrieve location information for an image or video.
- CamcorderProfile adds the QVGA (320x240) resolution profiles. Quality level is represented by the QUALITY_QVGA.and QUALITY_TIME_LAPSE_QVGAconstants.
- New methods setVideoStabilization(), getVideoStabilization(), and isVideoStabilizationSupported() let you check and manage video stabilization for a Camera.

18) Permissions

The following are new permissions:

- android.Manifest.permission#READ_SOCIAL_STREAM and android.Manifest.permission#WRITE_SOCIAL_STREAM: Allow a sync adapter to read and write social stream data to a contact in the shared Contacts Provider.

19)API Level

The Android 4.0.3 API is assigned an integer identifier—**15**—that is stored in the system itself. This identifier, called the "API level", allows the system to correctly determine whether an application is compatible with the system, prior to installing the application.

To use APIs introduced in Android 4.0.3 in your application, you need compile the application against an Android platform that supports API level 15 or higher. Depending on your needs, you might also need to add an android:minSdkVersion="15" attribute to the <uses-sdk> element.

19.1) Android 8.0 Features and APIs

Android 8.0 (API level 26) introduces a variety of new features and capabilities for users and developers.

Make sure to also check out Android 8.0 Behavior Changes to learn about areas where platform changes may affect your apps.

20) User Experience

21) Picture-in-Picture mode

Android 8.0 (API level 26) allows activities to launch inpicture-in-picture (PIP) mode. PIP is a special type of multi-window mode mostly used for video playback. PIP mode was originally available for Android TV only; Android 8.0 makes the feature available on other Android devices.

When an activity is in PIP mode, it is in the paused state, but should continue showing content. For this reason, you should make sure your app does not pause playback in its onPause() handler. Instead, you should pause video in onStop(), and resume playback in onStart().

To specify that your activity can use PIP mode, setandroid: supports Picture In Picture to true in the manifest. (Beginning with Android 8.0, PIP does not require the android: resizeableActivity manifest attribute. However, you must set android: resizeableActivity to `true` if your activity supports other multi-window modes.)

Android 8.0 (API level 26) introduces a new object, PictureInPictureParams, which you pass to PIP methods to specify how an activity should behave when it is in PIP mode. This object specifies properties such as the activity's preferred aspect ratio.

The existing PIP methods described in Adding Picture-in-picture can now be used on all Android devices, not just on Android TV. In addition, Android 8.0 provides the following methods to support PIP mode:

Activity. Enter Picture In Picture Mode(PictureInPictureParams args): Places the activity in picture-in-picture mode. The activity's aspect ratio and other configuration settings are specified by args. If any fields in args are empty, the system uses the values set the last time you called Activity.setPictureInPictureParams().

The specified activity is placed in a corner of the screen; the rest of the screen is filled with the previous activity that was on screen. The activity entering PIP mode goes into the paused state, but remains started. If the user taps the PIP activity, the system shows a menu for the user to interact with; no touch events reach the activity while it is in the PIP state.

- **Activity Set Picture In Picture Params():** Updates an activity's PIP configuration settings. If the activity is currently in PIP mode, the settings are updated; this is useful if activity's aspect ratio changes. If the activity is not in PIP mode, these configuration settings are used regardless of theenterPictureInPictureMode() method that you call.

22) Notifications

In Android 8.0 (API level 26), we've redesigned notifications to provide an easier and more consistent way to manage notification behavior and settings. These changes include:

Users can long-press on app launcher icons to view notifications in Android 8.0.

- **Notification channels**: Android 8.0 introduces notification channels that allow you to create a user-customizable channel for each type of notification you want to display. The user interface refers to notification channels as *notification categories*. To learn how to implement notification channels, see Managing notification channels.

- **Notification dots:** Android 8.0 introduces support for displaying dots, or badges, on app launcher icons. Notification dots reflect the presence of notifications that the user has not yet dismissed or acted on. To learn how to work with notification dots, seeNotification badges.
- **Snoozing:** Users can snooze notifications, which causes them to disappear for a period of time before reappearing. Notifications reappear with the same level of importance they first appeared with. Apps can remove or update a snoozed notification, but updating a snoozed notification does not cause it to reappear.
- **Notification timeouts:** You can set a timeout when creating a notification using setTimeoutAfter(). You can use this method to specify a duration after which a notification should be canceled. If required, you can cancel a notification before the specified timeout duration elapses.
- **Notification settings:** You can call setSettingsText() to set the text that appears when you create a link to your app's notification settings from a notification using the Notification.INTENT_CATEGORY_NOTIFICATION_PREFERENCES intent. The system may provide the following extras with the intent to filter the settings your app must display to users: EXTRA_CHANNEL_ID, NOTIFICATION_TAG and NOTIFICATION_ID.
- **Notification dismissal:** Users can dismiss notifications themselves, and apps can remove them programmatically. You can determine when a notification is dismissed and why it's dismissed by implementing the onNotificationRemoved() method from the NotificationListenerServiceclass.
- **Background colors:** You can set and enable a background color for a notification. You should only use this feature in notifications for ongoing tasks which are critical for a user to see at a glance. For example, you could set a background color for notifications related to driving directions, or a phone call in progress. You can also set the desired background color using setColor(). Doing so allows you to use setColorized() to enable the use of a background color for a notification.
- **Messaging style:** In Android 8.0, notifications that use the MessagingStyle class display more content in their collapsed form. You should use theMessagingStyle class for notifications that are messaging-related. You can also use

the addHistoricMessage() method to provide context to a conversation by adding historic messages to messaging-related notifications.

22.1) Autofill Framework

Account creation, login, and credit card transactions take time and are prone to errors. Users can easily get frustrated with apps that require these types of repetitive tasks.

Android 8.0 (API level 26) makes filling out forms, such as login and credit card forms, easier with the introduction of the Autofill Framework. Existing and new apps work with Autofill Framework after the user opts in to autofill.

You can take some steps to optimize how your app works with the framework.

22.2) Downloadable Fonts

Android 8.0 (API level 26) and Android Support Library 26 let you request fonts from a provider application instead of bundling fonts into the APK or letting the APK download fonts. This feature reduces your APK size, increases the app installation success rate, and allows multiple apps to share the same font.

22.3) Fonts in XML

Android 8.0 (API level 26) introduces a new feature, Fonts in XML, which lets you use fonts as resources. This means, there is no need to bundle fonts as assets. Fonts are compiled in R file and are automatically available in the system as a resource. You can then access these fonts with the help of a new resource type, font.

The Support Library 26 provides full support to this feature on devices running API versions 14 and higher.

22.4) Autosizing Text View

Android 8.0 (API level 26) lets you set the size of your text expand or contract automatically based on the size of the TextView. This means, it is much easier to optimize the text size on different screens or with dynamic content.

22.5) Adaptive icons

Android 8.0 (API level 26) introduces adaptive launcher icons. Adaptive icons support visual effects, and can display a variety of shapes across different device models. To learn how to create adaptive icons, see the Adaptive Icons guide.

22.6) Color Management

Android developers of imaging apps can now take advantage of new devices that have a wide-gamut color capable display. To display wide gamut images, apps will need to enable a flag in their manifest (per activity) and load bitmaps with an embedded wide color profile (AdobeRGB, Pro Photo RGB, DCI-P3, etc.).

22.7) Web View APIs

Android 8.0 provides several APIs to help you manage the WebView objects that display web content in your app. These APIs, which improve your app's stability and security, include the following:

- Version API
- Google SafeBrowsing API
- Termination Handle API
- Renderer Importance API

The WebView class now includes a Safe Browsing API to enhance the security of web browsing.

22.8) Pinning Shortcuts and Widgets

Android 8.0 (API level 26) introduces in-app pinning of shortcuts and widgets. In your app, you can create pinned shortcuts and widgets for supported launchers, subject to user permission.

22.9) Maximum Screen Aspect Ratio

Android 8.0 (API level 26) brings changes to how to configure an app's maximum aspect ratio.

First, Android 8.0 introduces the maxAspectRatio attribute, which you can use to set your app's maximum aspect ratio. In addition, in Android 8.0 and higher, an app's default maximum aspect ratio is the native aspect ratio of the device on which the app is running.

22.10) Multi-Display Support

Beginning with Android 8.0 (API level 26), the platform offers enhanced support for multiple displays. If an activity supports multi-window mode and is running on a device with multiple displays, users can move the activity from one display to another. When an app launches an activity, the app can specify which display the activity should run on.

Note: If an activity supports multi-window mode, Android 8.0 automatically enables multi-display support for that activity. You should test your app to make sure it works adequately in a multi-display environment.

Only one activity at a time can be in the resumed state, even if the app has multiple displays. The activity with focus is in the resumed state; all other visible activities are paused, but not stopped.

When a user moves an activity from one display to another, the system resizes the activity and issues runtime changes as necessary. Your activity can handle the configuration change itself, or it can allow the system to destroy the process containing your activity and recreate it with the new dimensions.

ActivityOptions provides two new methods to support multiple displays:

setLaunchDisplayId()

Specifies which display the activity should be shown on when it is launched.

getLaunchDisplayId()

Returns the activity's current launch display.

The adb shell is extended to support multiple displays. The shell start command can now be used to launch an activity, and to specify the activity's target display:

```
adb shell start <activity_name> --display <display_id>
```

23) Unified Layout Margins and Padding

Android 8.0 (API level 26) makes it easier for you to specify situations where opposite sides of a View element use the same margin or padding. Specifically, you can now use the following attributes in your layout XML files:

- layout_marginVertical, which
  defines layout_marginTop and layout_marginBottom at the same time.

- layout_marginHorizontal, which
  defines layout_marginLeft and layout_marginRight at the same time.
- paddingVertical, which defines paddingTop and paddingBottom at the same time.
- paddingHorizontal, which defines paddingLeft and paddingRight at the same time.

Note: If you customize your app's logic to support different languages and cultures, including text direction, keep in mind that these attributes don't affect the values of layout_marginStart, layout_marginEnd, paddingStart, or paddingEnd. You can set these values yourself, in addition to the new vertical and horizontal layout attributes, to create layout behavior that depends on the text direction.

24) Pointer Capture

Some apps, such as games, remote desktop, and virtualization clients, greatly benefit from getting control over the mouse pointer. Pointer capture is a new feature in Android 8.0 (API level 26) that provides such control by delivering all mouse events to a focused view in your app.

Starting in Android 8.0, a View in your app can request pointer capture and define a listener to process captured pointer events. The mouse pointer is hidden while in this mode. The view can release pointer capture when it doesn't need the mouse information anymore. The system can also release pointer capture when the view loses focus, for example, when the user opens another app.

25) App Categories

Android 8.0 (API level 26) allows each app to declare a category that it fits into, when relevant. These categories are used to cluster together apps of similar purpose or function when presenting them to users, such as in Data Usage, Battery Usage, or Storage Usage. You can define a category for your app by setting the android:appCategory attribute in your <application> manifest tag.

26) Android TV Launcher

Android 8.0 (API level 26) includes a new content-centric, Android TV home screen experience, which is available with the Android TV emulator and Nexus Player device image for Android 8.0. The new home screen organizes video content in rows corresponding to channels, which are each populated with programs by an app on the system. Apps can publish multiple channels, and users can configure which channels that they wish to see on the home screen. The Android TV home screen also includes a Watch Next row, which is populated with programs from apps, based on the viewing habits of the user. Apps can also provide video previews, which are automatically played when a user focuses on a program. The APIs for populating channels and programs are part of the TvProvider APIs, which are distributed as a Android Support Library module with Android 8.0.

27) Animator Set

Starting in Android 8.0 (API level 26), the AnimatorSet API now supports seeking and playing in reverse. Seeking lets you set the position of the animation set to a specific point in time. Playing in reverse is useful if your app includes animations for actions that can be undone. Instead of defining two separate animation sets, you can play the same one in reverse.

28) Input and Navigation

28.1) Keyboard Navigation Clusters

If an activity in your app uses a complex view hierarchy, such as the one in Figure 2, consider organizing groups of UI elements into clusters for easier keyboard navigation among them. Users can press Meta+Tab, or Search+Tab on Chromebook devices, to navigate from one cluster to another. Good examples of clusters include: side panels, navigation bars, main content areas, and elements that could contain many child elements.

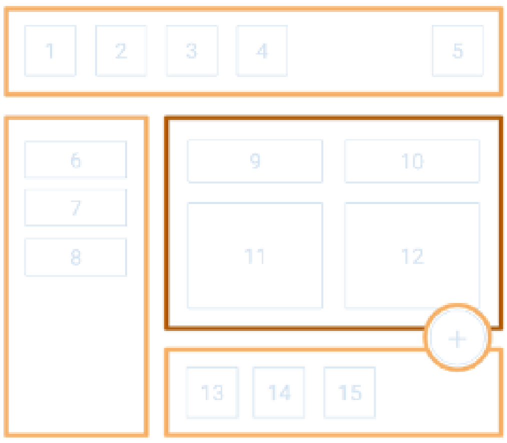


Figure 2. Activity containing 5 navigation clusters

To make a View or ViewGroup element a cluster, set theandroid: keyboardNavigationCluster attribute to true in the element's layout XML file, or pass true into setKeyboardNavigationCluster() in your app's UI logic.

Note: Clusters cannot be nested, although non-nested clusters may appear at different levels of the hierarchy. If you attempt to nest clusters, the framework treats only the top-most ViewGroup element as a cluster.

On devices that have touchscreens, you can set a cluster - designated ViewGroup object's android: touch screen Blocks Focus element to true to allow cluster-only navigation into and out of that cluster. If you apply this configuration to a cluster, users cannot use the Tab key or arrow keys to navigate into or out of the cluster; they must press the cluster navigation keyboard combination instead.

28.2) View Default Focus

In Android 8.0 (API level 26), you can assign the View that should receive focus after a (re)created activity is resumed and the user presses a keyboard navigation key, such as the tab key. To apply this "focused by default" setting, set a View element's android:focusedByDefault attribute to true in the layout XML file containing the UI element, or pass in true to setFocusedByDefault() in your app's UI logic.

28.3) Speech Output

Activities and services can use instances of TextToSpeech to dictate and pronounce content. As of Android 8.0 (API level 26), your app can obtain more precise timing information about when a text-to-speech engine begins speaking individual synthesized words, as long as the engine provides this information. You can use this functionality to call attention to specific words as the text-to-speech engine speaks them.

To use these text-to-speech engine improvements in your app, register an instance of UtteranceProgressListener. As part of the registration process, include a handler for the onRangeStart() method.

The text-to-speech engine calls rangeStart() to record the point in time at which it expects audio playback of a specific range of text to start. When the audio for that text range starts playback, your app's onRangeStart() method executes. Your app can then respond to this callback, such as by highlighting the text range that's associated with the utterance.

28.4) System

28.4.1) New Strict Mode Detectors

Android 8.0 (API level 26) adds three new StrictMode detectors to help identify potential bugs in your app:

- detectUnbufferedIo() will detect when your app reads or writes data without buffering, which can drastically impact performance.
- detectContentUriWithoutPermission() will detect when your app accidentally forgets to grant permissions to another app when starting an Activity outside your app.
- detectUntaggedSockets() will detect when your app performs network traffic without using setThreadStatsTag(int) to tag your traffic for debugging purposes.

28.4.2) Cached Data

Android 8.0 (API level 26) gives better guidance and behaviors around cached data. Each app is now given a disk space quota for cached data, as returned by getCacheQuotaBytes(UUID).

When the system needs to free up disk space, it will start by deleting cached files from apps that are the most over their allocated quota. Thus, if you keep your cached data under your allocated quota, your cached files will be some of the last on the system to be cleared when necessary. When the system is deciding what cached files to delete inside your app, it will consider the oldest files first (as determined by modified time).

There are also two new behaviors that you can enable on a per-directory basis to control how the system frees up your cached data:

- StorageManager.setCacheBehaviorAtomic() can be used to indicate that a directory and all of its contents should be deleted as a single atomic unit.
- setCacheBehaviorTombstone(File, boolean) can be used to indicate that instead of deleting files inside a directory, they should be truncated to be 0 bytes in length, leaving the empty file intact.

Finally, when you need to allocate disk space for large files, consider using the new allocateBytes(FileDescriptor, long) API, which will automatically clear cached files belonging to other apps (as needed) to meet your request. When deciding if the device has enough disk space to hold your new data, call getAllocatableBytes(UUID) instead of using getUsableSpace(), since the former will consider any cached data that the system is willing to clear on your behalf.

28.4.3) Content Provider Paging

We've updated content providers to include support for loading a large dataset one page at a time. For example, a photo app with many thousands of images can query for a subset of the data to present in a page. Each page of results returned by a content provider is represented by a single Cursor object. Both a client and a provider must implement paging to make use of this feature.

For detailed information about the changes to content providers, see ContentProvider and ContentProviderClient.

28.4.4) Content Refresh Requests

The ContentProvider and ContentResolver classes now each include a refresh() method, making it easier for clients to know whether the information they request is up-to-date.

You can add custom content refreshing logic by extending ContentProvider. Make sure that you override the refresh() method to return true, indicating to your provider's clients that you've attempted to refresh the data yourself.

Your client app can explicitly request refreshed content by calling a different method, also called refresh(). When calling this method, pass in the URI of the data to refresh.

Note: Because you may be requesting data over a network, you should invoke refresh() from the client side only when there's a strong indication that the content is stale. The most common reason to perform this type of content refresh is in response to a swipe-to-refresh gesture, explicitly requesting the current UI to display up-to-date content.

28.5) Job Scheduler Improvements

Android 8.0 (API level 26) introduces a number of improvements to JobScheduler. These improvements make it easier for your app to comply with the new background execution limits, since you can generally use scheduled jobs to replace the now-restricted background services or implicit broadcast receivers.

Updates to JobScheduler include:

- You can now associate a work queue with a scheduled job. To add a work item to a job's queue, call JobScheduler.enqueue(). When the job is running, it can take pending work off the queue and process it. This functionality handles many of the use cases that previously would have called for starting a background service, particulary services that implement IntentService.
- Android Support Library 26.0.0 introduces a new JobIntentService class, which provides the same functionality as IntentService but uses jobs instead of services when running on Android 8.0 (API level 26) or higher.
- You can now call JobInfo.Builder.setClipData() to associate a ClipData with a job. This option enables you to associate URI permission grants with a job, similarly to how these permissions can be propogated to Context.startService(). You can also use URI permission grants with intents on work queues.
- Scheduled jobs now support several new constraints:

JobInfo.isRequireStorageNotLow()

Job does not run if the device's available storage is low.

JobInfo.isRequireBatteryNotLow()

Job does not run if the battery level is at or below the criticial threshold; this is the level at which the device shows the **Low battery warning** system dialog.

29) NETWORK_TYPE_METERED

Job requires a metered network connection, like most cellular data plans.

29.1) Custom Data Store

Android 8.0 (API level 26) lets you provide a custom data store to your preferences, which can be useful if your app stores the preferences in a cloud or local database, or if the preferences are device-specific

30) Media Enhancements

30.1) Volume Shaper

There is a new VolumeShaper class. Use it to perform short automated volume transitions like fade-ins, fade-outs, and cross fades.

30.2) Audio Focus Enhancements

Audio apps share the audio output on a device by requesting and abandoning audio focus. An app handles changes in focus by starting or stopping playback, or ducking its volume. There is a new AudioFocusRequest class. Using this class as the parameter of requestAudioFocus(), apps have new capabilites when handling changes in audio focus: automatic ducking and delayed focus gain.

30.3) Media Metrics

A new getMetrics() method returns a PersistableBundle object containing configuration and performance information, expressed as a map of attributes and values. The getMetrics() method is defined for these media classes:

- MediaPlayer.getMetrics()
- MediaRecorder.getMetrics()
- MediaCodec.getMetrics()
- MediaExtractor.getMetrics()

Metrics are collected separately for each instance and persist for the lifetime of the instance. If no metrics are available the method returns null. The actual metrics returned depend on the class.

30.4) Media Player

Starting in Android 8.0 (API level 26) MediaPlayer can playback DRM-protected material and HLS sample-level encrypted media.

Android 8.0 introduces a new overloaded seekTo() command that provides fine-grained control when seeking to a frame. It includes a second parameter that specifies a seek mode:

- SEEK_PREVIOUS_SYNC moves the media position to a sync (or key) frame associated with a data source that is located right before or at the given time.

- SEEK_NEXT_SYNC moves the media position to a sync (or key) frame associated with a data source that is located right after or at the given time.
- SEEK_CLOSEST_SYNC moves the media position to a sync (or key) frame associated with a data source that is located closest to or at the given time.
- SEEK_CLOSEST moves the media position to a frame (*not necessarily a sync or key frame*) associated with a data source that is located closest to or at the given time.

When seeking continuously, apps should use any of the SEEK_ modes rather than SEEK_CLOSEST, which runs relatively slower but can be more precise.

30.5) Media Recorder

- MediaRecorder now supports the MPEG2_TS format which is useful for streaming:

```
mMediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.MPEG_2_
TS);
```

  see MediaRecorder.OutputFormat

- The MediaMuxer can now handle any number of audio and video streams. You are no longer limited to one audio track and/or one video track. Use addTrack() to mix as many tracks as you like.
- The MediaMuxer can also add one or more metadata tracks containing user-defined per-frame information. The format of the metadata is defined by your application. The metadata track is only supported for MP4 containers.

Metadata can be useful for offline processing. For example, gyro signals from the sensor could be used to perform video stabilization.

When adding a metadata track, the track's mime format must start with the prefix "application/". Writing metadata is the same as writing video/audio data except that the data does not come from a MediaCodec. Instead, the app passes a ByteBuffer with an associated timestamp to thewriteSampleData() method. The timestamp must be in the same time base as the video and audio tracks.

The generated MP4 file uses the TextMetaDataSampleEntry defined in section 12.3.3.2 of the ISOBMFF to signal the metadata's mime format. When using MediaExtractor to extract the file with metadata track, the mime format of the metadata will be extracted into MediaFormat.

31) Improved Media File Access

The Storage Access Framework (SAF) allows apps to expose a custom DocumentsProvider, which can provide access to files in a data source to other apps. In fact, a documents provider can even provide access to files that reside on network storage or that use a protocol like Media Transfer Protocol (MTP).

However, accessing large media files from a remote data source introduces some challenges:

- Media players require seekable access to a file from a documents provider. In cases where a large media file resides on a remote data source, the documents provider must fetch all of the data in advance and create a snapshot file descriptor. The media player cannot play the file without the file descriptor, thus playback cannot begin until the documents provider finishes downloading the file.
- Media collection managers, such as photo apps, must traverse a series of access URIs to reach media that's stored on an external SD card via scoped folders. This access pattern makes mass operations on media—such as moving, copying, and deleting—quite slow.
- Media collection managers cannot determine a document's location given its URI. This makes it difficult for these types of apps to allow users to choose where to save a media file.

Android 8.0 addresses each of these challenges by improving the Storage Access Framework.

32) Custom Document Providers

Starting in Android 8.0, the Storage Access Framework allows custom documents providers to create seekable file descriptors for files residing in a remote data source. The SAF can open a file to get a native seekable file descriptor. The SAF then delivers discrete bytes requests to the documents provider. This feature allows a documents provider to return the exact range of bytes that a media player app has requested instead of caching the entire file in advance.

To use this feature, you need to call the new StorageManager.openProxyFileDescriptor() method.
The openProxyFileDescriptor() method accepts a ProxyFileDescriptorCallback object as a callback. The SAF invokes the callback any time a client application performs file operations on the file descriptor returned from the documents provider.

33) Direct Document Access

As of Android 8.0 (API level 26), you can use the getDocumentUri() method to get a URI that references the same document as the given mediaUri. However, because the returned URI is backed by a DocumentsProvider, media collection managers can access the document directly, without having to traverse trees of scoped directories. As a result, the media managers can perform file operations on the document significantly more quickly.

33.1) Caution: The getDocumentUri() method only locates media files; it doesn't grant apps permission to access those files

34) Paths to Documents

When using the Storage Access Framework in Android 8.0 (API level 26), you can use the findDocumentPath() method, available in both the DocumentsContract and DocumentsProvider classes, to determine the path from the root of a file system given a document's ID. The method returns this path in a DocumentsContract.Path object. In cases where a file system has multiple defined paths to the same document, the method returns the path that is used most often to reach the document with the given ID.

This functionality is particularly useful in the following scenarios:

- Your app uses a "save as" dialog that displays the location of a particular document.
- Your app shows folders in a search results view and must load the child documents that are within a particular folder if the user selects that folder.

Note: If your app has permission to access only some of the documents in the path, the return value of findDocumentPath() includes only the folders and documents that your app can access.

35) Monitoring Audio Playback

The AudioManager system service maintains a list of active AudioPlaybackConfiguration objects, each of which contains information about a particular audio playback session. Your app can retrieve the set of currently-active configurations by calling getActivePlaybackConfigurations().

As of Android 8.0 (API level 26), you can register a callback that notifies your app when one or more AudioPlaybackConfiguration objects has changed. To do so, call registerAudioPlaybackCallback(), passing in an instance of AudioManager.AudioPlaybackCallback. TheAudioManager.AudioPlaybackCallback class contains the onPlaybackConfigChanged() method, which the system calls when the audio playback configuration changes.

36) Connectivity

36.1) Wi-Fi Aware

Android 8.0 (API level 26) adds support for Wi-Fi Aware, which is based on the Neighbor Awareness Networking (NAN) specification. On devices with the appropriate Wi-Fi Aware hardware, apps and nearby devices can discover and communicate over Wi-Fi without an Internet access point. We're working with our hardware partners to bring Wi-Fi Aware technology to devices as soon as possible. For information on how to integrate Wi-Fi Aware into your app, see Wi-Fi Aware.

36.2) Bluetooth

Android 8.0 (API level 26) enriches the platform's Bluetooth support by adding the following features:

- Support for the AVRCP 1.4 standard, which enables song-library browsing.
- Support for the Bluetooth Low-Energy (BLE) 5.0 standard.
- Integration of the Sony LDAC codec into the Bluetooth stack.

36.3) Companion Device Pairing

Android 8.0 (API level 26) provides APIs that allow you to customize the pairing request dialog when trying to pair with companion devices over Bluetooth, BLE, and Wi-Fi.

37) Sharing

37.1) Smart Sharing

Android 8.0 (API level 26) learns about users' personalized sharing preferences and better understands for each type of content which are the right apps to share with. For example, if a user takes a photo of a receipt, Android 8.0 can suggest an expense-tracking app; if the user takes a selfie, a social media app can better handle the image. Android 8.0 automatically learns all these patterns according to users' personalized preferences.

Smart sharing works for types of content other than image, such as audio, video, text, URL, etc.

To enable Smart sharing, add an ArrayList of up to three string annotations to the intent that shares the content. The annotations should describe the major components or topics in the content. The following code example shows how to add annotations to the intent:

```
ArrayList<String> annotations = new ArrayList<>();

annotations.add("topic1");
annotations.add("topic2");
annotations.add("topic3");

intent.putStringArrayListExtra(
    Intent.EXTRA_CONTENT_ANNOTATIONS,
    annotations
);
```

For detailed information about Smart sharing annotations, see EXTRA_CONTENT_ANNOTATIONS.

37.2) Text Classifier

On compatible devices, apps can use a new Text Classifier to check whether a string matches a known classifier entity type and get suggested selection alternatives. Entities recognized by the system include addresses, URLs, telephone numbers, and email addresses.

37.3) Accessibility

Android 8.0 (API level 26) supports several new accessibility features for developers who create their own accessibility services:

- A new volume category for adjusting accessibility volume.
- Fingerprint gestures as an input mechanism.
- Multilingual text to speech capabilities.
- A hardware-based accessibility shortcut for quickly accessing a preferred accessibility service.
- Support for continued gestures, or programmatic sequences of strokes.
- An accessibility button for invoking one of several enabled accessibility features (available only on devices that use a software-rendered navigation area).
- Standardized one-sided range values.
- Several features for processing text more easily, including hint text and locations of on-screen text characters.

38) Security & Privacy

38.1) Permissions

Android 8.0 (API level 26) introduces several new permissions related to telephony:

- The ANSWER_PHONE_CALLS permission allows your app to answer incoming phone calls programmatically. To handle an incoming phone call in your app, you can use the acceptRingingCall() method.
- The READ_PHONE_NUMBERS permission grants your app read access to the phone numbers stored in a device.

These permission are both classified as dangerous and are both part of the PHONE permission group.

38.2) New Account Access and Discovery APIs

Android 8.0 (API level 26) introduces several improvements to how apps get access to user accounts. For the accounts that they manage, authenticators can use their own policy to decide whether to hide accounts from, or reveal accounts to, an app. The Android system tracks applications which can access a particular account.

In previous versions of Android, apps that wanted to track the list of user accounts had to get updates about all accounts, including accounts with unrelated types. Android 8.0 adds the addOnAccountsUpdatedListener(android.accounts.OnAccountsUpdateListener, android.os.Handler, boolean, java.lang.String[]) method, which lets apps specify a list of account types for which account changes should be received.