# Android, the world's most popular mobile platform

Android powers hundreds of millions of mobile devices in more than 190 countries around the world. It's the largest installed base of any mobile platform and growing fast—every day another million users power up their Android devices for the first time and start looking for apps, games, and other digital content.

Android gives you a world-class platform for creating apps and games for Android users everywhere, as well as an open marketplace for distributing to them instantly.



Android growth in device activations

### Global partnerships and large installed base

Building on the contributions of the open-source Linux community and more than 300 hardware, software, and carrier partners, Android has rapidly become the fastest-growing mobile OS.

*Every day more than 1 million new Android devices are activated worldwide.*

Android’s openness has made it a favorite for consumers and developers alike, driving strong growth in app consumption. Android users download more than 1.5 billion apps and games from Google Play each month.

With its partners, Android is continuously pushing the boundaries of hardware and software forward to bring new capabilities to users and developers. For developers, Android innovation lets you build powerful, differentiated applications that use the latest mobile technologies.

### Powerful development framework

*Easily optimize a single binary for phones, tablets, and other devices.*

Android gives you everything you need to build best-in-class app experiences. It gives you a single application model that lets you deploy your apps broadly to hundreds of millions of users across a wide range of devices—from phones to tablets and beyond.

Android also gives you tools for creating apps that look great and take advantage of the hardware capabilities available on each device. It automatically adapts your UI to look its best on each device, while giving you as much control as you want over your UI on different device types.

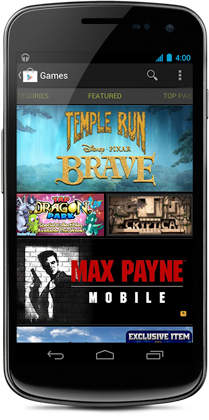
For example, you can create a single app binary that's optimized for both phone and tablet form factors. You declare your UI in lightweight sets of XML resources, one set for parts of the UI that are common to all form factors and other sets for optimzations specific to phones or tablets. At runtime, Android applies the correct resource sets based on its screen size, density, locale, and so on.

To help you develop efficiently, the [Android Developer Tools](http://developer.android.com/tools/index.html) offer a full Java IDE with advanced features for developing, debugging, and packaging Android apps. Using the IDE, you can develop on any available Android device or create virtual devices that emulate any hardware configuration.

*1.5 billion downloads a month and growing. Get your apps in front of millions of users at Google's scale.*

### Open marketplace for distributing your apps

Google Play is the premier marketplace for selling and distributing Android apps. When you publish an app on Google Play, you reach the huge installed base of Android.



As an open marketplace, Google Play puts you in control of how you sell your products. You can publish whenever you want, as often as you want, and to the customers you want. You can distribute broadly to all markets and devices or focus on specific segments, devices, or ranges of hardware capabilities.

You can monetize in the way that works best for your business—priced or free, with in-app products or subscriptions—for highest engagement and revenues. You also have complete control of the pricing for your apps and in-app products and can set or change prices in any supported currency at any time.

Beyond growing your customer base, Google Play helps you build visibility and engagement across your apps and brand. As your apps rise in popularity, Google Play gives them higher placement in weekly "top" charts and rankings, and for the best apps promotional slots in curated collections.

Preinstalled on hundreds of millions of Android devices around the world, Google Play can be a growth engine for your business.

**Get Started**

Everything you need to start developing apps for Android is available here on developer.android.com. You'll find everything from the developer SDK, API documentation, and design guidelines, to information about the current device landscape and how you can distribute and monetize your app.

No two apps are built in the same way, but we've structured the information you need to build an app into the following three sections that represent the general order for app development.

**1. Design**

Before you write a single line of code, you need to design the user interface and make it fit the Android user experience. Although you may know what a user will *do*with your app, you should pause to focus on how a user will *interact*with it. Your design should be sleek, simple, powerful, and tailored to the Android experience.

So whether you're a one-man shop or a large team, you should study the Design guidelines first.

**2. Develop**

Once your design is finalized, all you need are the tools to turn your app ideas into reality. Android's framework provides you the APIs to build apps that take full advantage of device hardware, connected accessory devices, the Internet, software features, and more. With the power of Android, there's no limit to the power of your apps.

Everything you need to learn about the app framework and developer tools is in the Develop documentation.

**3. Distribute**

Now your app is complete. You've built it to support a variety of screen sizes and densities, and tested it on the Android emulator and on real devices. You're ready to ship your app.

How you proceed depends on a variety of factors, such as your monetization strategy and which types of devices your app supports. Everything you need to get started with this process is available in the Distribute section.

Now that you know what's available, get started by installing the Android SDK.

**Android Design**



Welcome to **Android Design**, your place for learning how to design exceptional Android apps.

# Creative Vision



We focused the design of Android around three overarching goals, which apply to our core apps as well as the system at large. As you design apps to work with Android, consider these goals:

#### Enchant me

Beauty is more than skin deep. Android apps are sleek and aesthetically pleasing on multiple levels. Transitions are fast and clear; layout and typography are crisp and meaningful. App icons are works of art in their own right. Just like a well-made tool, your app should strive to combine beauty, simplicity and purpose to create a magical experience that is effortless and powerful.

#### Simplify my life

Android apps make life easier and are easy to understand. When people use your app for the first time, they should intuitively grasp the most important features. The design work doesn't stop at the first use, though. Android apps remove ongoing chores like file management and syncing. Simple tasks never require complex procedures, and complex tasks are tailored to the human hand and mind. People of all ages and cultures feel firmly in control, and are never overwhelmed by too many choices or irrelevant flash.

#### Make me amazing

It's not enough to make an app that is easy to use. Android apps empower people to try new things and to use apps in inventive new ways. Android lets people combine applications into new workflows through multitasking, notifications, and sharing across apps. At the same time, your app should feel personal, giving people access to superb technology with clarity and grace.

 Design Principles

These design principles were developed by and for the Android User Experience Team to keep users' best interests in mind. Consider them as you apply your own creativity and design thinking. Deviate with purpose.

## Enchant Me

#### Delight me in surprising ways

A beautiful surface, a carefully-placed animation, or a well-timed sound effect is a joy to experience. Subtle effects contribute to a feeling of effortlessness and a sense that a powerful force is at hand.



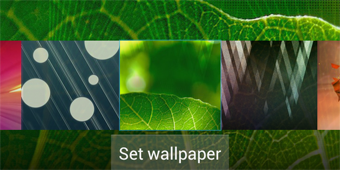
#### Real objects are more fun than buttons and menus

Allow people to directly touch and manipulate objects in your app. It reduces the cognitive effort needed to perform a task while making it more emotionally satisfying.



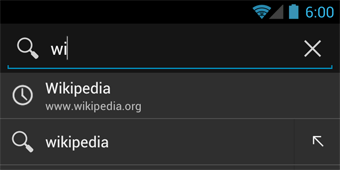
#### Let me make it mine

People love to add personal touches because it helps them feel at home and in control. Provide sensible, beautiful defaults, but also consider fun, optional customizations that don't hinder primary tasks.



#### Get to know me

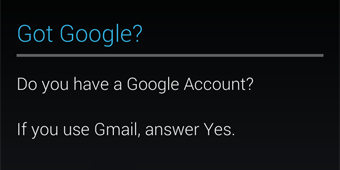
Learn peoples' preferences over time. Rather than asking them to make the same choices over and over, place previous choices within easy reach.



## Simplify My Life

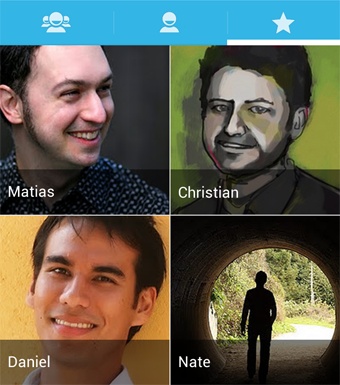
#### Keep it brief

Use short phrases with simple words. People are likely to skip sentences if they're long.



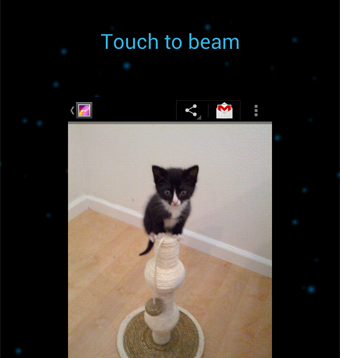
#### Pictures are faster than words

Consider using pictures to explain ideas. They get people's attention and can be much more efficient than words.



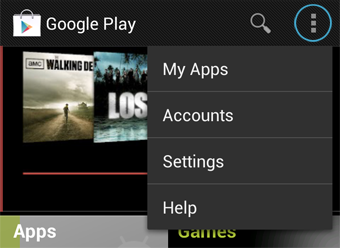
#### Decide for me but let me have the final say

Take your best guess and act rather than asking first. Too many choices and decisions make people unhappy. Just in case you get it wrong, allow for 'undo'.



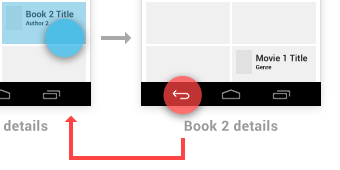
#### Only show what I need when I need it

People get overwhelmed when they see too much at once. Break tasks and information into small, digestible chunks. Hide options that aren't essential at the moment, and teach people as they go.



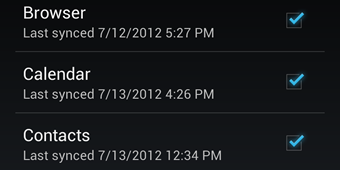
#### I should always know where I am

Give people confidence that they know their way around. Make places in your app look distinct and use transitions to show relationships among screens. Provide feedback on tasks in progress.



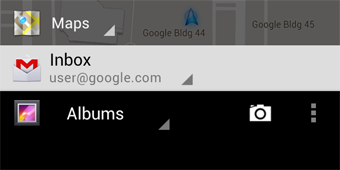
#### Never lose my stuff

Save what people took time to create and let them access it from anywhere. Remember settings, personal touches, and creations across phones, tablets, and computers. It makes upgrading the easiest thing in the world.



#### If it looks the same, it should act the same

Help people discern functional differences by making them visually distinct rather than subtle. Avoid modes, which are places that look similar but act differently on the same input.



#### Only interrupt me if it's important

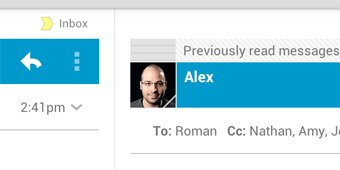
Like a good personal assistant, shield people from unimportant minutiae. People want to stay focused, and unless it's critical and time-sensitive, an interruption can be taxing and frustrating.



## Make Me Amazing

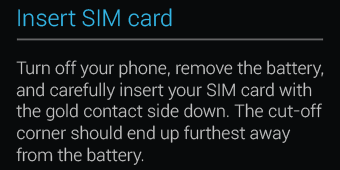
#### Give me tricks that work everywhere

People feel great when they figure things out for themselves. Make your app easier to learn by leveraging visual patterns and muscle memory from other Android apps. For example, the swipe gesture may be a good navigational shortcut.



#### It's not my fault

Be gentle in how you prompt people to make corrections. They want to feel smart when they use your app. If something goes wrong, give clear recovery instructions but spare them the technical details. If you can fix it behind the scenes, even better.



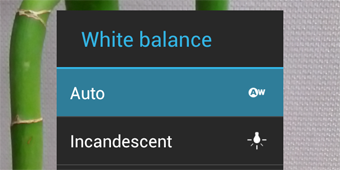
#### Sprinkle encouragement

Break complex tasks into smaller steps that can be easily accomplished. Give feedback on actions, even if it's just a subtle glow.



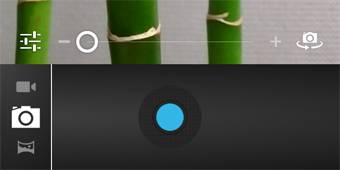
#### Do the heavy lifting for me

Make novices feel like experts by enabling them to do things they never thought they could. For example, shortcuts that combine multiple photo effects can make amateur photographs look amazing in only a few steps.



#### Make important things fast

Not all actions are equal. Decide what's most important in your app and make it easy to find and fast to use, like the shutter button in a camera, or the pause button in a music player.



# UI Overview

Android's system UI provides the framework on top of which you build your app. Important aspects include the Home screen experience, global device navigation, and notifications.

Your app will play an important part in keeping the overall Android experience consistent and enjoyable to use. At the end of this chapter we introduce the main elements for achieving this goal in your app.

Read on for a quick overview of the most important aspects of the Android user interface.

## Home, All Apps, and Recents



#### Home screen

Home is a customizable space that houses app shortcuts, folders and widgets. Navigate between different home screen panels by swiping left and right.

The Favorites Tray at the bottom always keeps your most important shortcuts and folders in view regardless of which panel is currently showing.

Access the entire collection of apps and widgets by touching the All Apps button at the center of the Favorites Tray.



#### All apps screen

The All Apps screen lets you browse the entire set of apps and widgets that are installed on your device.

Users can drag an app or widget icon from the All Apps screen and place it in any empty location on any Home screen.



#### Recents screen

Recents provides an efficient way of switching between recently used applications. It provides a clear navigation path between multiple ongoing tasks.

The Recents button at the right side of the navigation bar displays the apps that the user has interacted with most recently. They are organized in reverse chronological order with the most recently used app at the bottom.

Switch to an app by touching it. Remove an item by swiping left or right.

## System Bars

The system bars are screen areas dedicated to the display of notifications, communication of device status, and device navigation. Typically the system bars are displayed concurrently with your app. Apps that display immersive content, such as movies or images, can temporarily hide the system bars to allow the user to enjoy full screen content without distraction.



#### Status Bar

Displays pending notifications on the left and status, such as time, battery level, or signal strength, on the right. Swipe down from the status bar to show notification details.

#### Navigation Bar

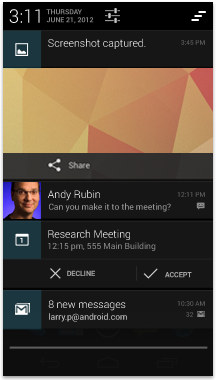
New for phones in Android 4.0, the navigation bar is present only on devices that don't have the traditional hardware keys. It houses the device navigation controls Back, Home, and Recents, and also displays a menu for apps written for Android 2.3 or earlier.

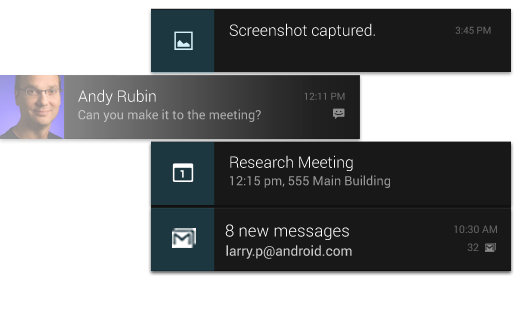
#### Combined Bar

On tablet form factors the status and navigation bars are combined into a single bar at the bottom of the screen.

## Notifications

Notifications are brief messages that users can access at any time from the status bar. They provide updates, reminders, or information that's important, but not critical enough to warrant interrupting the user. Open the notifications drawer by swiping down on the status bar. Touching a notification opens the associated app. [More on Notifications](http://developer.android.com/design/patterns/notifications.html)

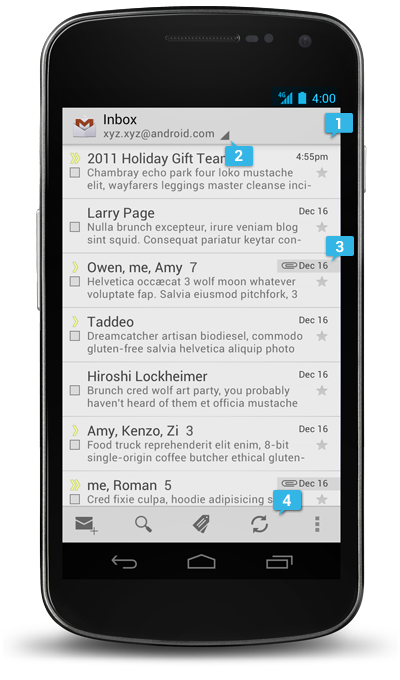




Notifications can be expanded to uncover more details and relevant actions. When collapsed, notifications have a one-line title and a one-line message.The recommended layout for a notification includes two lines. If necessary, you can add a third line.

Swiping a notification right or left removes it from the notification drawer.

## Common App UI



A typical Android app consists of action bars and the app content area.

#### Main Action Bar

The command and control center for your app. The main action bar includes elements for navigating your app's hierarchy and views, and also surfaces the most important actions.

[More on the Action Bar](http://developer.android.com/design/patterns/actionbar.html)

#### View Control

Allows users to switch between the different views that your app provides. Views typically consist of different arrangements of your data or different functional aspects of your app.

#### Content Area

The space where the content of your app is displayed.

#### Split Action Bar

Split action bars provide a way to distribute actions across additional bars located below the main action bar or at the bottom of the screen. In this example, a split action bar moves important actions that won't fit in the main bar to the bottom.

# UI Overview

Android's system UI provides the framework on top of which you build your app. Important aspects include the Home screen experience, global device navigation, and notifications.

Your app will play an important part in keeping the overall Android experience consistent and enjoyable to use. At the end of this chapter we introduce the main elements for achieving this goal in your app.

Read on for a quick overview of the most important aspects of the Android user interface.

## Home, All Apps, and Recents



#### Home screen

Home is a customizable space that houses app shortcuts, folders and widgets. Navigate between different home screen panels by swiping left and right.

The Favorites Tray at the bottom always keeps your most important shortcuts and folders in view regardless of which panel is currently showing.

Access the entire collection of apps and widgets by touching the All Apps button at the center of the Favorites Tray.



#### All apps screen

The All Apps screen lets you browse the entire set of apps and widgets that are installed on your device.

Users can drag an app or widget icon from the All Apps screen and place it in any empty location on any Home screen.



#### Recents screen

Recents provides an efficient way of switching between recently used applications. It provides a clear navigation path between multiple ongoing tasks.

The Recents button at the right side of the navigation bar displays the apps that the user has interacted with most recently. They are organized in reverse chronological order with the most recently used app at the bottom.

Switch to an app by touching it. Remove an item by swiping left or right.

## System Bars

The system bars are screen areas dedicated to the display of notifications, communication of device status, and device navigation. Typically the system bars are displayed concurrently with your app. Apps that display immersive content, such as movies or images, can temporarily hide the system bars to allow the user to enjoy full screen content without distraction.



#### Status Bar

Displays pending notifications on the left and status, such as time, battery level, or signal strength, on the right. Swipe down from the status bar to show notification details.

#### Navigation Bar

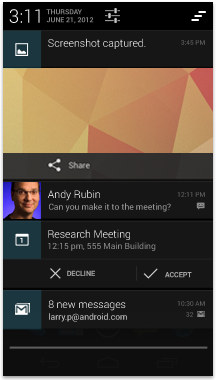
New for phones in Android 4.0, the navigation bar is present only on devices that don't have the traditional hardware keys. It houses the device navigation controls Back, Home, and Recents, and also displays a menu for apps written for Android 2.3 or earlier.

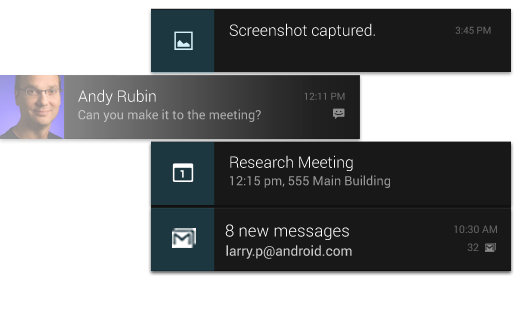
#### Combined Bar

On tablet form factors the status and navigation bars are combined into a single bar at the bottom of the screen.

## Notifications

Notifications are brief messages that users can access at any time from the status bar. They provide updates, reminders, or information that's important, but not critical enough to warrant interrupting the user. Open the notifications drawer by swiping down on the status bar. Touching a notification opens the associated app. [More on Notifications](http://developer.android.com/design/patterns/notifications.html)

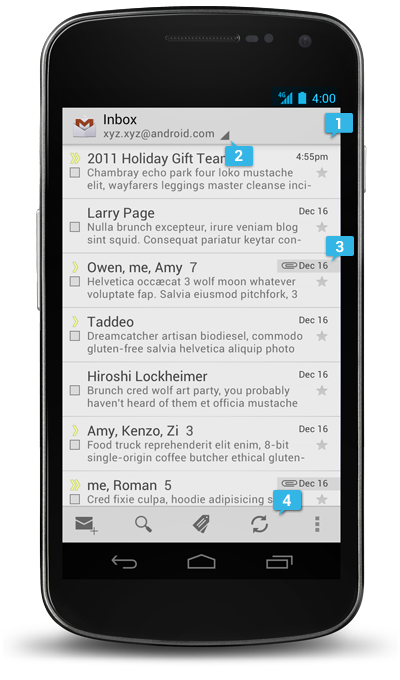




Notifications can be expanded to uncover more details and relevant actions. When collapsed, notifications have a one-line title and a one-line message.The recommended layout for a notification includes two lines. If necessary, you can add a third line.

Swiping a notification right or left removes it from the notification drawer.

## Common App UI



A typical Android app consists of action bars and the app content area.

#### Main Action Bar

The command and control center for your app. The main action bar includes elements for navigating your app's hierarchy and views, and also surfaces the most important actions.

[More on the Action Bar](http://developer.android.com/design/patterns/actionbar.html)

#### View Control

Allows users to switch between the different views that your app provides. Views typically consist of different arrangements of your data or different functional aspects of your app.

#### Content Area

The space where the content of your app is displayed.

#### Split Action Bar

Split action bars provide a way to distribute actions across additional bars located below the main action bar or at the bottom of the screen. In this example, a split action bar moves important actions that won't fit in the main bar to the bottom.

# 

# Jelly Bean

[](http://developer.android.com/images/jb-android-43@2x.png)

Welcome to Android 4.3, a sweeter version of Jelly Bean!

Android 4.3 includes performance optimizations and great new features for users and developers. This document provides a glimpse of what's new for developers.

See the Android 4.3 APIs document for a detailed look at the new developer APIs.

## Faster, Smoother, More Responsive

Android 4.3 builds on the performance improvements already included in Jelly Bean — **vsync timing**, **triple buffering**, **reduced touch latency**, **CPU input boost**, and**hardware-accelerated 2D rendering** — and adds new optimizations that make Android even faster.

For a graphics performance boost, the hardware-accelerated 2D renderer now **optimizes the stream of drawing commands**, transforming it into a more efficient GPU format by rearranging and merging draw operations. For multithreaded processing, the renderer can also now use **multithreading across multiple CPU cores** to perform certain tasks.

Android 4.3 also improves **rendering for shapes and text**. Shapes such as circles and rounded rectangles are now rendered at higher quality in a more efficient manner. Optimizations for text include increased performance when using multiple fonts or complex glyph sets (CJK), higher rendering quality when scaling text, and faster rendering of drop shadows.

**Improved window buffer allocation** results in a faster image buffer allocation for your apps, reducing the time taken to start rendering when you create a window.

For highest-performance graphics, Android 4.3 introduces support for **OpenGL ES 3.0** and makes it accessible to apps through both framework and native APIs. On supported devices, the hardware accelerated 2D rendering engine takes advantage of OpenGL ES 3.0 to optimize **texture management** and increase **gradient rendering fidelity**.

## OpenGL ES 3.0 for High-Performance Graphics

Android 4.3 introduces platform support for Khronos OpenGL ES 3.0, providing games and other apps with highest-performance 2D and 3D graphics capabilities on supported devices. You can take advantage of OpenGL ES 3.0 and related EGL extensions using either **framework APIs** or **native API bindings** through the Android Native Development Kit (NDK).

Key new functionality provided in OpenGL ES 3.0 includes acceleration of advanced visual effects, high quality ETC2/EAC texture compression as a standard feature, a new version of the GLSL ES shading language with integer and 32-bit floating point support, advanced texture rendering, and standardized texture size and render-buffer formats.

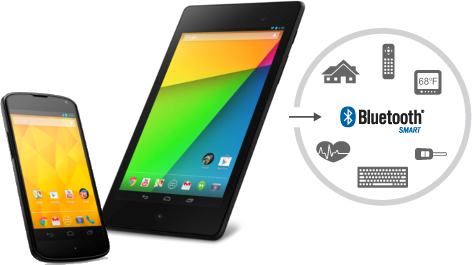
You can use the OpenGL ES 3.0 APIs to create highly complex, highly efficient graphics that run across a range of compatible Android devices, and you can support a single, standard texture-compression format across those devices.

OpenGL ES 3.0 is an optional feature that depends on underlying graphics hardware. Support is already available on Nexus 7 (2013), Nexus 4, and Nexus 10 devices.

## Enhanced Bluetooth Connectivity

#### Connectivity with Bluetooth Smart devices and sensors

Now you can design and build apps that interact with the latest generation of small, low-power devices and sensors that use Bluetooth Smart technology.



Android 4.3 gives you a single, standard API for interacting with Bluetooth Smart devices.

Android 4.3 introduces built-in platform support for **Bluetooth Smart Ready** in the central role and provides a standard set of APIs that apps can use to discover nearby devices, query for GATT services, and read/write characteristics.

With the new APIs, your apps can efficiently scan for devices and services of interest. For each device, you can check for supported GATT services by UUID and manage connections by device ID and signal strength. You can connect to a GATT server hosted on the device and read or write characteristics, or register a listener to receive notifications whenever those characteristics change.

You can implement support for any GATT profile. You can read or write standard characteristics or add support for custom characteristics as needed. Your app can function as either client or server and can transmit and receive data in either mode. The APIs are generic, so you’ll be able to support interactions with a variety of devices such as proximity tags, watches, fitness meters, game controllers, remote controls, health devices, and more.

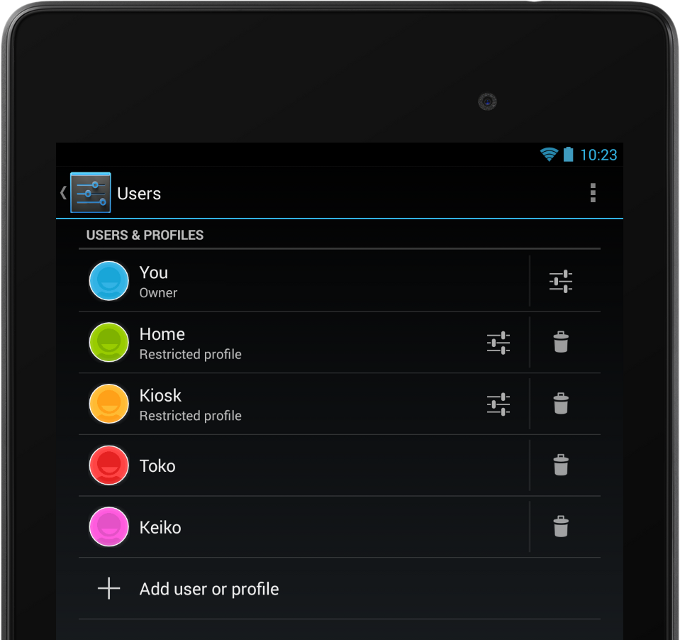
Support for Bluetooth Smart Ready is already available on Nexus 7 (2013) and Nexus 4 devices and will be supported in a growing number of Android-compatible devices in the months ahead.

#### AVRCP 1.3 Profile

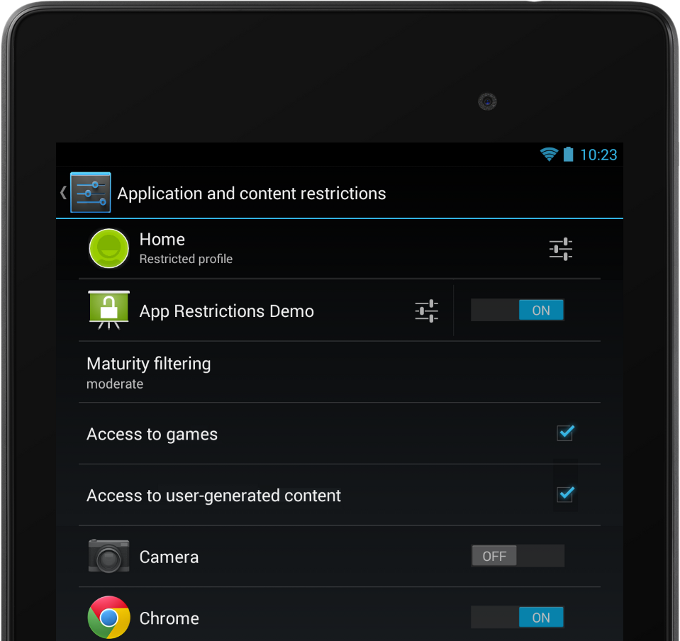
Android 4.3 adds built-in support for **Bluetooth AVRCP 1.3**, so your apps can support richer interactions with remote streaming media devices. Apps such as media players can take advantage of AVRCP 1.3 through the **remote control client APIs** introduced in Android 4.0. In addition to exposing playback controls on the remote devices connected over Bluetooth, apps can now transmit metadata such as track name, composer, and other types of media metadata.

Platform support for AVRCP 1.3 is built on the Bluedroid Bluetooth stack introduced by Google and Broadcom in Android 4.2. Support is available right away on Nexus devices and other Android-compatible devices that offer A2DP/AVRCP capability.

## Support for Restricted Profiles



A tablet owner can set up one or more restricted profiles in Settings and manage them independently.



Your app can offer restrictions to let owners manage your app content when it's running in a profile.

Android 4.3 extends the multiuser feature for tablets with**restricted profiles**, a new way to manage users and their capabilities on a single device. With restricted profiles, tablet owners can quickly set up **separate environments** for each user, with the ability to manage **finer-grained restrictions** in the apps that are available in those environments. Restricted profiles are ideal for friends and family, guest users, kiosks, point-of-sale devices, and more.

Each restricted profile offers an isolated and secure space with its own local storage, home screens, widgets, and settings. Unlike with users, profiles are created from the tablet owner’s environment, based on the owner’s installed apps and system accounts. The owner controls which installed apps are enabled in the new profile, and access to the owner’s accounts is disabled by default.

Apps that need to access the owner’s accounts — for sign-in, preferences, or other uses — can opt-in by declaring a manifest attribute, and the owner can review and manage those apps from the profile configuration settings.

For developers, restricted profiles offer a new way to deliver more value and control to your users. You can implement **app restrictions** — content or capabilities controls that are supported by your app — and advertise them to tablet owners in the profile configuration settings.

You can add app restrictions directly to the profile configuration settings using predefined boolean, select, and multi-select types. If you want more flexibility, you can even launch your own UI from profile configuration settings to offer any type of restriction you want.

When your app runs in a profile, it can check for any restrictions configured by the owner and enforce them appropriately. For example, a media app might offer a restriction to let the owner set a maturity level for the profile. At run time, the app could check for the maturity setting and then manage content according to the preferred maturity level.

If your app is not designed for use in restricted profiles, you can opt out altogether, so that your app can't be enabled in any restricted profile.

## Optimized Location and Sensor Capabilities

Google Play services offers advanced location APIs that you can use in your apps. Android 4.3 **optimizes these APIs** on supported devices with new hardware and software capabilities that minimize use of the battery.



**Hardware geofencing** optimizes for power efficiency by performing location computation in the device hardware, rather than in software. On devices that support hardware geofencing, Google Play services geofence APIs will be able to take advantage of this optimization to save battery while the device is moving.

**Wi-Fi scan-only mode** is a new platform optimization that lets users keep Wi-Fi scan on without connecting to a Wi-Fi network, to improve location accuracy while conserving battery. Apps that depend on Wi-Fi for location services can now ask users to enable scan-only mode from Wi-Fi advanced settings. Wi-Fi scan-only mode is not dependent on device hardware and is available as part of the Android 4.3 platform.

New sensor types allow apps to better manage sensor readings. A **game rotation vector** lets game developers sense the device’s rotation without having to worry about magnetic interference. **Uncalibrated gyroscope** and **uncalibrated magnetometer** sensors report raw measurements as well as estimated biases to apps.

The new hardware capabilities are already available on Nexus 7 (2013) and Nexus 4 devices, and any device manufacturer or chipset vendor can build them into their devices.

## New Media Capabilities

#### Modular DRM framework

To meet the needs of the next generation of media services, Android 4.3 introduces a **modular DRM framework** that enables media application developers to more easily integrate DRM into their own streaming protocols, such as MPEG DASH (Dynamic Adaptive Streaming over HTTP, ISO/IEC 23009-1).

Through a combination of new APIs and enhancements to existing APIs, the media DRM framework provides an**integrated set of services** for managing licensing and provisioning, accessing low-level codecs, and decoding encrypted media data. A new MediaExtractor API lets you get the PSSH metadata for DASH media. Apps using the media DRM framework manage the network communication with a license server and handle the streaming of encrypted data from a content library.

#### VP8 encoder

Android 4.3 introduces built-in support for **VP8 encoding**, accessible from framework and native APIs. For apps using native APIs, the platform includes **OpenMAX 1.1.2 extension headers** to support VP8 profiles and levels. VP8 encoding support includes settings for target bitrate, rate control, frame rate, token partitioning, error resilience, reconstruction and loop filters. The platform API introduces VP8 encoder support in a range of formats, so you can take advantage of the best format for your content.

VP8 encoding is available in software on all compatible devices running Android 4.3. For highest performance, the platform also supports hardware-accelerated VP8 encoding on capable devices.

#### Video encoding from a surface

Starting in Android 4.3 you can use a surface as the input to a video encoder. For example, you can now direct a stream from an OpenGL ES surface to the encoder, rather than having to copy between buffers.

#### Media muxer

Apps can use new media muxer APIs to combine elementary audio and video streams into a single output file. Currently apps can multiplex a single MPEG-4 audio stream and a single MPEG-4 video stream into a **single MPEG-4 ouput file**. The new APIs are a counterpart to the media demuxing APIs introduced in Android 4.2.

#### Playback progress and scrubbing in remote control clients

Since Android 4.0, media players and similar applications have been able to offer playback controls from remote control clients such as the device lock screen, notifications, and remote devices connected over Bluetooth. Starting in Android 4.3, those applications can now also expose playback **progress and speed** through their remote control clients, and receive commands to jump to a specific **playback position**.

## New Ways to Build Beautiful Apps

### Access to notifications

Notifications have long been a popular Android feature because they let users see information and updates from across the system, all in one place. Now in Android 4.3, apps can **observe the stream of notifications** with the user's permission and display the notifications in any way they want, including sending them to nearby devices connected over Bluetooth.

You can access notifications through new APIs that let you **register a notification listener** service and with permission of the user, receive notifications as they are displayed in the status bar. Notifications are delivered to you in full, with all details on the originating app, the post time, the content view and style, and priority. You can evaluate fields of interest in the notifications, process or add context from your app, and route them for display in any way you choose.

The new API gives you callbacks when a notification is added, updated, and removed (either because the user dismissed it or the originating app withdrew it). You'll be able to launch any intents attached to the notification or its actions, as well as dismiss it from the system, allowing your app to provide a complete user interface to notifications.

**Users remain in control** of which apps can receive notifications. At any time, they can look in Settings to see which apps have notification access and **enable or disable access** as needed. Notification access is disabled by default — apps can use a new Intent to take the user directly to the Settings to enable the listener service after installation.

#### View overlays

You can now create **transparent overlays** on top of Views and ViewGroups to render a temporary View hierarchy or transient animation effects without disturbing the underlying layout hierarchy. Overlays are particularly useful when you want to create animations such as sliding a view outside of its container or dragging items on the screen without affecting the view hierarchy.

#### Optical bounds layout mode

A new layout mode lets you manage the positioning of Views inside ViewGroups according to their **optical bounds**, rather than their clip bounds. Clip bounds represent a widget’s actual outer boundary, while the new optical bounds describe the where the widget appears to be, within the clip bounds. You can use the optical bounds layout mode to properly align widgets that use outer visual effects such as shadows and glows.

#### Custom rotation animation types

Apps can now define the exit and entry animation types used on a window when the device is rotated. You can set window properties to enable **jump-cut**, **cross-fade**, or **standard** window rotation. The system uses the custom animation types when the window is fullscreen and is not covered by other windows.

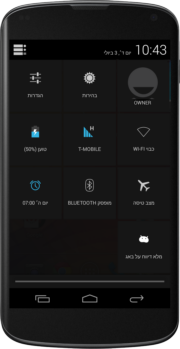
#### Screen orientation modes

Apps can set new orientation modes for Activities to ensure that they are displayed in the proper orientation when the device is flipped. Additionally, apps can use a new mode to **lock the screen** to its current orientation. This is useful for apps using the camera that want to **disable rotation** while shooting video.

#### Intent for handling Quick Responses

Android 4.3 introduces a new public Intent that lets any app **handle Quick Responses** — text messages sent by the user in response to an incoming call, without needing to pick up the call or unlock the device. Your app can listen for the intent and send the message to the caller over your messaging system. The intent includes the recipient (caller) as well as the message itself.

## Support for International Users

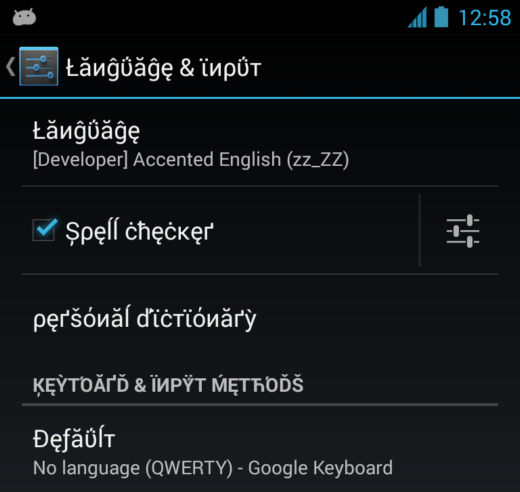
More parts of Android 4.3 are optimized for RTL languages.

#### RTL improvements

Android 4.3 includes RTL performance enhancements and broader RTL support across framework UI widgets, including ProgressBar/Spinner and ExpandableListView. More debugging information visible through the uiautomatorviewer tool. In addition, more system UI components are now RTL aware, such as notifications, navigation bar and the Action Bar.

To provide a better systemwide experience in RTL scripts, more default system apps now support RTL layouts, including Launcher, Quick Settings, Phone, People, SetupWizard, Clock, Downloads, and more.

#### Utilities for localization



Pseudo-locales make it easier to test your app's localization.

Android 4.3 also includes new utilities and APIs for creating better RTL strings and testing your localized UIs. A new **BidiFormatter** provides a set of simple APIs for wrapping Unicode strings so that you can fine-tune your text rendering in RTL scripts. To let you use this utility more broadly in your apps, the BidiFormatter APIs are also now available for earlier platform versions through the Support Package in the Android SDK.

To assist you with managing date formatting across locales, Android 4.3 includes a new **getBestDateTimePattern()** method that automatically generates the best possible localized form of a Unicode UTS date for a locale that you specify. It’s a convenient way to provide a more localized experience for your users.

To help you test your app more easily in other locales, Android 4.3 introduces **pseudo-locales** as a new developer option. Pseudo-locales simulate the language, script, and display characteristics associated with a locale or language group. Currently, you can test with a pseudo-locale for **Accented English**, which lets you see how your UI works with script accents and characters used in a variety of European languages.

## Accessibility and UI Automation

Starting in Android 4.3, accessibility services can **observe and filter key events**, such as to handle keyboard shortcuts or provide navigation parity with gesture-based input. The service receives the events and can process them as needed before they are passed to the system or other installed apps.

Accessibility services can declare **new capability attributes** to describe what their services can do and what platform features they use. For example, they can declare the capability to filter key events, retrieve window content, enable explore-by-touch, or enable web accessibility features. In some cases, services must declare a capability attribute before they can access related platform features. The system uses the service’s capability attributes to generate an opt-in dialog for users, so they can see and agree to the capabilities before launch.

Building on the accessibility framework in Android 4.3, a new **UI automation framework** lets tests interact with the device’s UI by simulating user actions and introspecting the screen content. Through the UI automation framework you can perform basic operations, set rotation of the screen, generate input events, take screenshots, and much more. It’s a powerful way to automate testing in realistic user scenarios, including actions or sequences that span multiple apps.

## Enterprise and Security

#### Wi-Fi configuration for WPA2-Enterprise networks

Apps can now configure the **Wi-Fi credentials** they need for connections to **WPA2 enterprise access points**. Developers can use new APIs to configure Extensible Authentication Protocol (EAP) and Encapsulated EAP (Phase 2) credentials for authentication methods used in the enterprise. Apps with permission to access and change Wi-Fi can configure authentication credentials for a variety of EAP and Phase 2 authentication methods.

#### Android sandbox reinforced with SELinux

Android now uses **SELinux**, a mandatory access control (MAC) system in the Linux kernel to augment the UID based application sandbox. This protects the operating system against potential security vulnerabilities.

#### KeyChain enhancements

The KeyChain API now provides a method that allows applications to confirm that system-wide keys are bound to a**hardware root of trust** for the device. This provides a place to create or store private keys that **cannot be exported** off the device, even in the event of a root or kernel compromise.

#### Android Keystore Provider

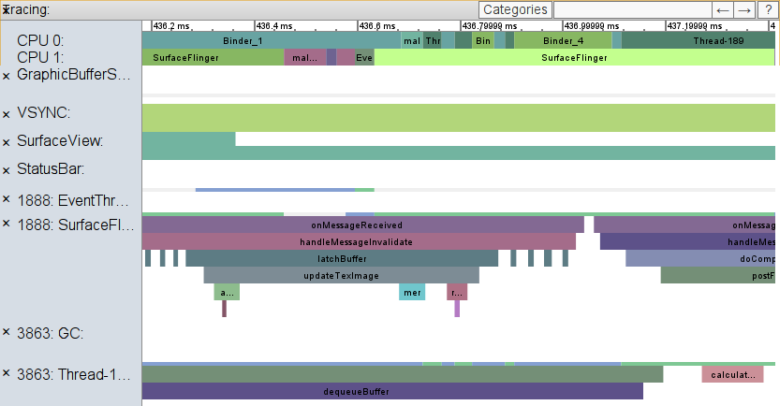
Android 4.3 introduces a keystore provider and APIs that allow applications to create exclusive-use keys. Using the APIs, apps can create or store private keys that **cannot be seen or used by other apps**, and can be added to the keystore without any user interaction.

The keystore provider provides the same security benefits that the KeyChain API provides for system-wide credentials, such as binding credentials to a device. Private keys in the keystore cannot be exported off the device.

#### Restrict Setuid from Android Apps

The /system partition is now mounted nosuid for zygote-spawned processes, preventing Android applications from executing setuid programs. This reduces root attack surface and likelihood of potential security vulnerabilities.

## New Ways to Analyze Performance



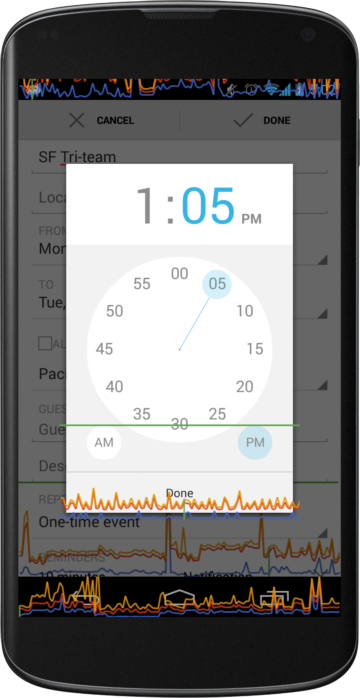
Systrace uses a new command syntax and lets you collect more types of profiling data.

#### Enhanced Systrace logging

Android 4.3 supports an enhanced version of the**Systrace** tool that’s easier to use and that gives you access to more types of information to profile the performance of your app. You can now collect trace data from **hardware modules**, **kernel functions**,**Dalvik VM** including garbage collection, **resources loading**, and more.

Android 4.3 also includes new Trace APIs that you can use in your apps to mark specific sections of code to trace using Systrace **begin/end events**. When the marked sections of code execute, the system writes the begin/end events to the trace log. There's minimal impact on the performance of your app, so timings reported give you an accurate view of what your app is doing.

You can visualize app-specific events in a timeline in the Systrace output file and analyze the events in the context of other kernel and user space trace data. Together with existing Systrace tags, custom app sections can give you new ways to understand the performance and behavior of your apps.

On-screen GPU profiling in Android 4.3.

#### On-screen GPU profiling

Android 4.3 adds new developer options to help you analyze your app’s performance and pinpoint rendering issues on any device or emulator.

In the **Profile GPU rendering** option you can now visualize your app’s effective framerate on-screen, while the app is running. You can choose to display profiling data as on-screen **bar or line graphs**, with colors indicating time spent creating drawing commands (blue), issuing the commands (orange), and waiting for the commands to complete (yellow). The system updates the on-screen graphs continuously, displaying a graph for each visible Activity, including the navigation bar and notification bar.

A green line highlights the **60ms threshold** for rendering operations, so you can assess the your app’s effective framerate relative to a 60 fps goal. If you see operations that cross the green line, you can analyze them further using Systrace and other tools.

On devices running Android 4.2 and higher, developer options are hidden by default. You can reveal them at any time by tapping 7 times on **Settings > About phone > Build number** on any compatible Android device.

#### StrictMode warning for file URIs

The latest addition to the StrictMode tool is a policy constraint that warns when your app exposes a file:// URI to the system or another app. In some cases the receiving app may not have access to the file:// URI path, so when sharing files between apps, a content:// URI should be used (with the appropriate permission). This new policy helps you catch and fix such cases. If you’re looking for a convenient way to store and expose files to other apps, try using theFileProvider content provider that’s available in the Support Library.

# Jelly Bean



Welcome to Android 4.2, the latest version ofJelly Bean!

Android 4.2 has performance optimizations, a refreshed system UI, and great new features for users and developers. This document provides a glimpse of what's new for developers.

See the Android 4.2 APIs document for a detailed look at the new developer APIs.

## Faster, Smoother, More Responsive

Android 4.2 builds on the performance improvements already included in Jelly Bean —**vsync timing**, **triple buffering**, **reduced touch latency**, and **CPU input boost** — and adds new optimizations that make Android even faster.

Improvements in the **hardware-accelerated 2D renderer** make common animations such as scrolling and swiping smoother and faster. In particular, **drawing is optimized** for layers, clipping and certain shapes (rounded rects, circles and ovals).

A variety of **WebView rendering optimizations** make scrolling of web pages smoother and free from jitter and lags.

Android’s **Renderscript Compute** is the first computation platform ported to run directly on a **mobile device GPU**. It automatically takes advantage of **GPU computation** resources whenever possible, dramatically improving performance for graphics and image processing. Any app using Renderscript on a supported device can benefit immediately from this GPU integration **without recompiling**.



## Refined, refreshed UI

Android 4.2 refines the Jelly Bean user experience and brings familiar Android UI patterns such as status bar, system bar, and notifications window to all tablets.

All screen sizes now feature the **status bar** on top, with pull-down access to **notifications** and a new **Quick Settings** menu. The familiar system bar appears on the bottom, with buttons easily accessible from either hand. The **Application Tray** is also available on all screen sizes.

## One tablet, many users

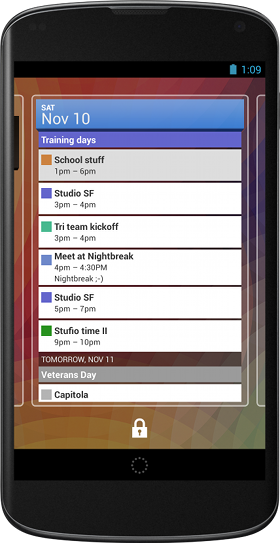
Now several users can **share a single Android tablet**, with each user having convenient access to a **dedicated user space**. Users can switch to their spaces with a single touch from the lock screen.

On a multiuser device, Android gives each user a separate environment, including user-specific emulated SD card storage. Users also have their own homescreens, widgets, accounts, settings, files, and apps, and the system keeps these separate. All users share core system services, but the system ensures that each user's applications and data remain isolated. In effect, each of the multiple users has his or her own Android device.

Users can install and uninstall apps at any time in their own environments. To save storage space, Google Play downloads an APK only if it's not already installed by another user on the device. If the app is already installed, Google Play records the new user's installation in the usual way but doesn't download another copy of the app. Multiple users can run the same copy of an APK because the system creates a new instance for each user, including a user-specific data directory.

For developers, **multi-user support is transparent** — your apps do not need to do anything special to run normally in a multi-user environment and there are no changes you need to make in your existing or published APKs. The system manages your app in each user space just as it does in a single-user environment.

## New ways to engage users



You can extend **app widgets** to run on the lock screen, for instant access to your content.

### Lock screen widgets

In Android 4.2, users can place **app widgets** directly on their **lock screens**, for instant access to favorite app content without having to unlock. Users can add as many as five lock screen widgets, choosing from widgets provided by installed apps. The lock screen displays each widget in its own panel, letting users swipe left and right to view different panels and their widgets.

Like all app widgets, lock screen widgets can display **any kind of content**and they can accept direct user interaction. They can be entirely self-contained, such as a widget that offers controls to play music, or they can let users jump straight to an Activity in your app, after unlocking along the way as needed.

For developers, lock screen widgets offer a great new way to engage users. They let you put your content in front of users in a location they’ll see often, and they give you more opportunities to bring users directly into your app.

You can take advantage of this new capability by building a new app widget or by extending an existing home screen widget. If your app already includes home screen widgets, you can extend them to the lock screen with minimal change. To give users an optimal experience, you can update the widget to use the full lock screen area when available and resize when needed on smaller screens. You can also add features to your widgets that might be especially useful or convenient on the lock screen.

### Daydream

Daydream is an **interactive screensaver mode** that starts when a user’s device is docked or charging. In this mode, the system launches a daydream — a remote content service provided by an installed app — as the device screensaver. A user can enable Daydream from the Settings app and then choose the daydream to display.

Daydreams combine the best capabilities of live wallpapers and home screen widgets, but they are more powerful. They let you offer the any kind of content in a completely new context, with user interactions such as flipping through photos, playing audio or video, or jumping straight into your app with a single touch.

Because daydreams can start automatically when a device is charging or docked, they also give your app a great way to support new types of user experiences, such as leanback or exhibition mode, demo or kiosk mode, and "attract mode" — all without requiring special hardware.



Daydream lets you create powerful interactive screensavers that display any kind of content.

Daydreams are similar to Activities and can do anything that Activity can do — from rendering a UI hierarchy (without using RemoteViews) to drawing directly using Canvas, OpenGL, SurfaceTexture, and more. They can play video and audio and they can even accept direct user interaction. However, daydreams are not Activities, so they don’t affect the backstack or appear in Recents and they cannot be launched directly from your app.

Implementing a daydream is straightforward and you can take advantage of UI components and resources that you’ve already created for other parts of your app. You can provide multiple daydreams in your app and you can offer distinct content and display settings for each.

## External display support

Android 4.2 introduces platform support for **external displays** that goes far beyond mirroring — apps can now target unique content to any one or multiple displays that are attached to an Android device. Apps can build on this to deliver new kinds of interaction and entertainment experiences to users.

### Display manager

Apps interact with displays through a new display manager system service. Your app can enumerate the displays and check the capabilities of each, including size, density, display name, ID, support for secure video, and more. Your app can also receive callbacks when displays are added or removed or when their capabilities change, to better manage your content on external displays.

### Presentation window

To make it easy to show content on an external display, the framework provides a new UI object called a **Presentation** — a type of dialog that represents a window for your app’s content on a specific external display. Your app just gives the display to use, a theme for the window, and any unique content to show. The Presentation handles inflating resources and rendering your content according to the characteristics of the targeted display.



You can take full control of two or more independent displays using **Presentation**.

A Presentation gives your app full control over the remote display window and its content and lets you manage it based on user input events such as key presses, gestures, motion events, and more. You can use all of the normal tools to create a UI and render content in the Presentation, from building an arbitrary view hierarchy to using SurfaceView or SurfaceTexture to draw directly into the window for streamed content or camera previews.

### Preferred display selection

When multiple external displays are available, you can create as many Presentations as you need, with each one showing unique content on a specific display. In many cases, you might only want to show your content on a single external display — but always on the that’s best for Presentation content. For this, the system can help your app choose the best display to use.

To find the best display to use, your app can query the display manager for the system’s **preferred Presentation display**and receive callbacks when that display changes. Alternatively, you can use the media router service, extended in Android 4.2, to receive notifications when a system video route changes. Your app can display content by default in the main Activity until a preferred Presentation display is attached, at which time it can automatically switch to Presentation content on the preferred display. Your apps can also use media router’s MediaRouteActionProvider and MediaRouteButton to offer standard display-selection UI.

### Protected content

For apps that handle protected or encrypted content, the display API now reports the **secure video capabilities** of attached displays. Your app query a display to find out if it offers a secure video output or provides protected graphics buffers and then choose the appropriate content stream or decoding to make the content viewable. For additional security on SurfaceView objects, your app can set a secure flag to indicate that the contents should never appear in screenshots or on a non-secure display output, even when mirrored.

### Wireless display

Starting in Android 4.2, users on supported devices can connect to an external display over Wi-Fi, using Miracast, a peer-to-peer wireless display standard created by the Wi-Fi Alliance. When a wireless display is connected, users can stream any type of content to the big screen, including photos, games, maps, and more.

Apps can take advantage of **wireless displays** in the same way as they do other external displays and no extra work is needed. The system manages the network connection and streams your Presentation or other app content to the wireless display as needed.

## Native RTL support



Developers can now **mirror their layouts** for RTL languages.

Android 4.2 introduces **full native support for RTL** (right-to-left) layouts, including layout mirroring. With native RTL support, you can deliver the same great app experience to all of your users, whether their language uses a script that reads right-to-left or one that reads left-to-right.

When the user switches the system language to a right-to-left script, the system now provides automatic mirroring of app UI layouts and all view widgets, in addition to bidi mirroring of text elements for both reading and character input.

Your app can take advantage of **RTL layout mirroring** in your app with minimal effort. If you want the app to be mirrored, you simply declare a new attribute in your app manifest and change all "left/right" layout properties to new "start/end" equivalents. The system then handles the mirroring and display of your UI as appropriate.

For precise control over your app UI, Android 4.2 includes new APIs that let you manage layout direction, text direction, text alignment, gravity, and locale direction in View components. You can even create custom versions of layout, drawables, and other resources for display when a right-to-left script is in use.

To help you debug and optimize your custom right-to-left layouts, the HierarchyViewer tool now lets you see start/end properties, layout direction, text direction, and text alignment for all the Views in the hierarchy.

## Enhancements for international languages

Android 4.2 includes a variety of **font and character optimizations** for international users:

* For Korean users, a new font choice is available — Nanum (나눔글꼴) Gothic, a unicode font designed especially for the Korean-language script.
* Improved support for Japanese vertical text displayed in WebViews.
* Improved font kerning and positioning for Indic, Thai, Arabic, and Hebrew default fonts.

The default Android keyboard also includes an updated set of dictionaries:

* Improved dictionaries for French (with bigram support), English, and Russian
* New dictionaries for Danish, Greek, Finnish, Lithuanian, Latvian, Polish, Slovenian, Serbian, Swedish, Turkish

## New ways to create beautiful UI

### Nested Fragments

For more control over your UI components and to make them more modular, Android 4.2 lets you **nest Fragments inside of Fragments**. For any Fragment, a new Fragment manager lets you insert other Fragments as child nodes in the View hierarchy.

You can use nested Fragments in a variety of ways, but they are especially useful for implementing dynamic and reusable UI components inside of a UI component that is itself dynamic and reusable. For example, if you use ViewPager to create fragments that swipe left and right, you can now insert fragments into each Fragment of the view pager.

To let you take advantage of nested Fragments more broadly in your app, this capability is added to the latest version of the **Android Support Library**.

## Accessibility

The system now helps accessibility services **distinguish between touch exploration and accessibility gestures** while in touch-exploration mode. When a user touches the screen, the system notifies the service that a generic touch interaction has started. It then tracks the speed of the touch interaction and determines whether it is a touch exploration (slow) or accessibility gesture (fast) and notifies the service. When the touch interaction ends, the system notifies the service.

The system provides a new global accessibility option that lets an accessibility service open the Quick Settings menu based on an action by the user. Also added in Android 4.2 is a new accessibility feedback type for **Braille devices**.

To give accessibility services insight into the meaning of Views for accessibility purposes, the framework provides new APIs for associating a View as the label for another View. The label for each View is available to accessibility services through AccessibilityNodeInfo.

## Improved Camera with HDR

Android 4.2 introduces a **new camera hardware interface and pipeline** for improved performance. On supported devices, apps can use a new **HDR camera scene mode** to capture an image using high dynamic range imaging techniques.

Additionally, the framework now provides an API to let apps check whether the camera shutter sound can be disabled. Apps can then let the user disable the sound or choose an alternative sound in place of the standard shutter sound, which is recommended.

## Renderscript Computation

In Android 4.2, Renderscript Compute introduces new scripting features, new optimizations, and direct GPU integration for the highest performance in computation operations.

### Filterscript

Filterscript is a subset of Renderscript that is focused on **optimized image processing across a broad range of device chipsets**. Developers can write their image processing operations in Filterscript using the standard Renderscript runtime API, but within stricter constraints that ensure wider compatibility and improved optimization across CPUs, GPUs, and DSPs.

Filterscript is ideal for hardware-accelerating simple image-processing and computation operations such as those that might be written for OpenGL ES fragment shaders. Because it places a relaxed set of constraints on hardware, your operations are optimized and accelerated on more types of device chipsets. Any app targeting API level 17 or higher can make use of Filterscript.

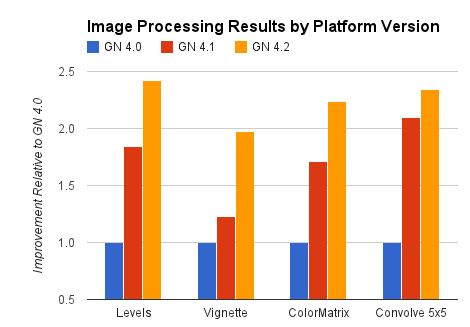
### Script intrinsics

In Android 4.2, Renderscript adds support for a set of script intrinsics — pre-implemented **filtering primitives that are accelerated** to reduce the amount of code that you need to write and to ensure that your app gets the maximum performance gain possible.

Intrinsics are available for blends, blur, color matrix, 3x3 and 5x5 convolve, per-channel lookup table, and converting an Android YUV buffer to RGB.

### Script groups

You can now create **groups of Renderscript scripts** and execute them all with a single call as though they were part of a single script. This allows Renderscript to optimize execution of the scripts in ways that it could not do if the scripts were executed individually.



Renderscript image-processing benchmarks run on different Android platform versions (Android 4.0, 4.1, and 4.2) in CPU only on a Galaxy Nexus device.

If you have a directed acyclic graph of Renderscript operations to run, you can use a builder class to create a script group defining the operations. At execution time, Renderscript optimizes the run order and the connections between these operations for best performance.

### Ongoing optimization improvements

When you use Renderscript for computation operations, you apps benefit from **ongoing performance and optimization improvements** in the Renderscript engine itself, without any impact on your app code or any need for recompilation.

As optimization improves, your operations execute faster and on more chipsets, without any work on your part. The chart at right highlights the performance gain delivered by ongoing Renderscript optimization improvements across successive versions of the Android platform.

### GPU Compute

Renderscript Compute is the first computation platform ported to run directly on a mobile device GPU. It now automatically takes advantage of **GPU computation** resources whenver possible to improve performance. With GPU integration, even the most complex computations for graphics or image processing can execute with dramatically improved performance.

Any app using Renderscript on a supported device can benefit immediately from this GPU integration, without recompiling. The Nexus 10 tablet is the first device to support this integration.

## New built-in developer options

The Android 4.2 system includes a variety of new developer options that make it easier to create great looking apps that perform well. The new options expose features for **debugging and profiling** your app from any device or emulator.

On devices running Android 4.2, developer options are hidden by default, helping to create a better experience for users. You can reveal the developer options at any time by tapping 7 times on **Settings** > **About phone** > **Build number** on any compatible Android device.



New developer options give you more ways to profile and debug on a device.

New developer options in Android 4.2 include:

* **Take bug report** — immediately takes a screen shot and dumps device state information to local file storage, then attaches them to a new outgoing email message.
* **Power menu bug reports** — Adds a new option to the device power menu and quick settings to take a bug report (see above).
* **Verify apps over usb** — Allows you to disable app checks for sideloading apps over USB, while still checking apps from other sources like the browser. This can speed up the development process while keeping the security feature enabled.
* **Show hardware layers updates** — Flashes hardware layers green when they update.
* **Show GPU overdraw** — Highlights GPU overdraw areas.
* **Force 4x MSAA** — Enables 4x MSAA in Open GL ES 2.0 apps.
* **Simulate secondary displays** — Creates one or more non-secure overlay windows on the current screen for use as a simulated remote display. You can control the simulated display’s size and density.
* **Enable OpenGL traces** — Lets you trace OpenGL execution using Logcat, Systrace, or callstack on glGetError.

## New Platform Technologies

Android 4.2 includes a variety of new and **enhanced platform technologies** to support innovative communications use-cases across a broad range of hardware devices. In most cases, the new platform technologies and enhancements do not directly affect your apps, so you can benefit from them without any modification.

### Security enhancements

Every Android release includes dozens of security enhancements to protect users. Here are some of the enhancements in Android 4.2:

* **Application verification** — Users can choose to enable “Verify Apps" and have applications screened by an application verifier, prior to installation. App verification can alert the user if they try to install an app that might be harmful; if an application is especially bad, it can block installation.
* **More control of premium SMS** — Android will provide a notification if an application attempts to send SMS to a short code that uses premium services which might cause additional charges. The user can choose whether to allow the application to send the message or block it.
* **Always-on VPN** — VPN can be configured so that applications will not have access to the network until a VPN connection is established. This prevents applications from sending data across other networks.
* **Certificate Pinning** — The libcore SSL implementation now supports certificate pinning. Pinned domains will receive a certificate validation failure if the certificate does not chain to a set of expected certificates. This protects against possible compromise of Certificate Authorities.
* **Improved display of Android permissions** — Permissions have been organized into groups that are more easily understood by users. During review of the permissions, the user can click on the permission to see more detailed information about the permission.
* **installd hardening** — The installd daemon does not run as the root user, reducing potential attack surface for root privilege escalation.
* **init script hardening** — init scripts now apply O\_NOFOLLOW semantics to prevent symlink related attacks.
* **FORTIFY\_SOURCE** — Android now implements FORTIFY\_SOURCE. This is used by system libraries and applications to prevent memory corruption.
* **ContentProvider default configuration** — Applications which target API level 17 will have “export” set to “false” by default for each ContentProvider, reducing default attack surface for applications.
* **Cryptography** — Modified the default implementations of SecureRandom and Cipher.RSA to use OpenSSL. Added SSLSocket support for TLSv1.1 and TLSv1.2 using OpenSSL 1.0.1
* **Security Fixes** — Upgraded open source libraries with security fixes include WebKit, libpng, OpenSSL, and LibXML. Android 4.2 also includes fixes for Android-specific vulnerabilities. Information about these vulnerabilities has been provided to Open Handset Alliance members and fixes are available in Android Open Source Project. To improve security, some devices with earlier versions of Android may also include these fixes.

### New Bluetooth stack

Android 4.2 introduces a new Bluetooth stack optimized for use with Android devices. The new Bluetooth stack developed in collaboration between Google and Broadcom replaces the stack based on BlueZ and provides improved compatibility and reliability.

### Low-latency audio

Android 4.2 improves support for low-latency audio playback, starting from the improvements made in Android 4.1 release for audio output latency using OpenSL ES, Soundpool and tone generator APIs. These improvements depend on hardware support — devices that offer these low-latency audio features can advertise their support to apps through a hardware feature constant. New AudioManager APIs are provided to query the native audio sample rate and buffer size, for use on devices which claim this feature.

### New camera hardware interface

Android 4.2 introduces a new implementation of the camera stack. The camera subsystem includes the implementations for components in the camera pipeline such as burst mode capture with processing controls.

### New NFC hardware interface and controller interface

Android 4.2 introduces support for controllers based on the NCI standard from the NFC-Forum. NCI provides a standard communication protocol between an NFC Controller (NFCC) and a device Host, and the new NFC stack developed in collaboration between Google and Broadcom supports it.

### Dalvik runtime optimizations

The Dalvik runtime includes enhancements for performance and security across a wider range of architectures:

* x86 JIT support from Intel and MIPS JIT support from MIPS
* Optimized garbage-collection parameters for devices with > 512MB
* Default implementations of SecureRandom and Cipher.RSA now use OpenSSL
* SSLSocket support for TLSv1.1 and TLSv1.2 via OpenSSL 1.0.1
* New intrinsic support for StrictMath methods abs, min, max, and sqrt
* BouncyCastle updated to 1.47
* zlib updated to 1.27
* dlmalloc updated to 2.8.6

# Jelly Bean



Welcome to Android 4.1 the first version of Jelly Bean!

Android 4.1 is the fastest and smoothest version of Android yet. We’ve made improvements throughout the platform and added great new features for users and developers. This document provides a glimpse of what's new for developers.

See the Android 4.1 APIs document for a detailed look at the new developer APIs.

## Faster, Smoother, More Responsive

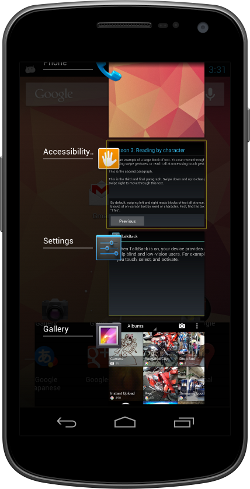
Android 4.1 is optimized to deliver Android's best performance and lowest touch latency, in an effortless, intuitive UI.

To ensure a consistent framerate, Android 4.1 extends **vsync timing** across all drawing and animation done by the Android framework. Everything runs in lockstep against a 16 millisecond vsync heartbeat — application rendering, touch events, screen composition, and display refresh — so frames don’t get ahead or behind.

Android 4.1 also adds **triple buffering** in the graphics pipeline, for more consistent rendering that makes everything feel smoother, from scrolling to paging and animations.

Android 4.1 reduces touch latency not only by **synchronizing touch** to vsync timing, but also by actually **anticipating**where your finger will be at the time of the screen refresh. This results in a more reactive and uniform touch response. In addition, after periods of inactivity, Android applies a **CPU input boost** at the next touch event, to make sure there’s no latency.

**Tooling** can help you get the absolute best performance out of your apps. Android 4.1 is designed to work with a new tool called **systrace**, which collects data directly from the Linux kernel to produce an overall picture of system activities. The data is represented as a group of vertically stacked time series graphs, to help isolate rendering interruptions and other issues. The tool is available now in the Android SDK (Tools R20 or higher)



## Enhanced Accessibility

New APIs for accessibility services let you handle gestures and manage **accessibility focus** as the user moves through the on-screen elements and navigation buttons using accessibility gestures, accessories, and other input. The Talkback system and explore-by-touch are redesigned to use accessibility focus for easier use and offer a complete set of APIs for developers.

Accessibility services can link their own **tutorials** into the Accessibility settings, to help users configure and use their services.

Apps that use standard View components **inherit support** for the new accessibility features automatically, without any changes in their code. Apps that use custom Views can use new accessibility node APIs to indicate the parts of the View that are of interest to accessibility services.

## Support for International Users



### Bi-Directional Text and Other Language Support

Android 4.1 helps you to reach more users through support for **bi-directional text** in TextView and EditText elements. Apps can display text or handle text editing in left-to-right or right-to-left scripts. Apps can make use of new Arabic and Hebrew locales and associated fonts.

Other types of new language support include:

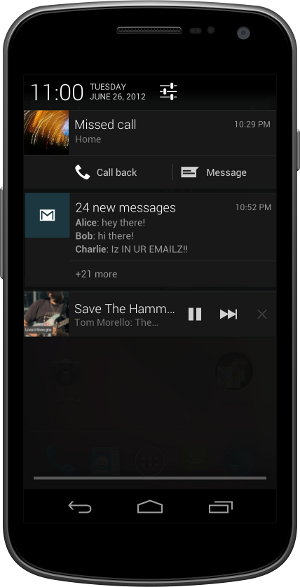
* Additional Indic languages: Kannada, Telugu, and Malayalam
* The new Emoji characters from Unicode version 6.0
* Better glyph support for Japanese users (renders Japanese-specific versions of glyphs when system language is set to Japanese)
* Arabic glyphs optimized for WebViews in addition to the Arabic glyphs for TextViews
* Vertical Text support in WebViews, including Ruby Text and additional Vertical Text glyphs
* Synthetic Bold is now available for all fonts that don't have dedicated bold glyphs

### User-installable keymaps

The platform now supports **user-installable keyboard maps**, such as for additional international keyboards and special layout types. By default, Android 4.1 includes 27 international keymaps for keyboards, including Dvorak. When users connect a keyboard, they can go to the Settings app and select one or more keymaps that they want to use for that keyboard. When typing, users can switch between keymaps using a shortcut (ctrl-space).

You can create an app to **publish additional keymaps** to the system. The APK would include the keyboard layout resources in it, based on standard Android keymap format. The application can offer additional keyboard layouts to the user by declaring a suitable broadcast receiver for ACTION\_QUERY\_KEYBOARD\_LAYOUTS in its manifest.

## New Ways to Create Beautiful UI



Developers can create custom notification styles like those shown in the examples above to display rich content and actions.

### Expandable notifications

Notifications have long been a unique and popular feature on Android. Developers can use them to place important or time-based information in front of users in the notification bar, outside of the app’s normal UI.

Android 4.1 brings a major update to the Android notifications framework. Apps can now display **larger, richer notifications** to users that can be expanded and collapsed with a pinch or swipe. Notifications support **new types of content**, including photos, have configurable priority, and can even include multiple actions.

Through an improved **notification builder**, apps can create notifications that use a larger area, up to 256 dp in height. Three **templated notification styles** are available:

* BigTextStyle — a notification that includes a multiline TextView object.
* BigInboxStyle — a notification that shows any kind of list such as messages, headlines, and so on.
* BigPictureStyle — a notification that showcases visual content such as a bitmap.

In addition to the templated styles, you can create your own notification styles **using any remote View**.

Apps can add up to three **actions** to a notification, which are displayed below the notification content. The actions let the users respond directly to the information in the notification in alternative ways. such as by email or by phone call, without visiting the app.

With expandable notifications, apps can give more information to the user, effortlessly and on demand. Users remain in control and can long-press any notification to get information about the sender and optionally disable further notifications from the app.



App Widgets can resize automatically to fit the home screen and load different content as their sizes change.

### Resizable app widgets

Android 4.1 introduces improved App Widgets that can **automatically resize**, based on where the user drops them on the home screen, the size to which the user expands them, and the amount of room available on the home screen. New App Widget APIs let you take advantage of this to**optimize your app widget content** as the size of widgets changes.

When a widget changes size, the system notifies the host app’s widget provider, which can reload the content in the widget as needed. For example, a widget could display larger, richer graphics or additional functionality or options. Developers can still maintain control over maximum and minimum sizes and can update other widget options whenever needed.

You can also supply separate landscape and portrait layouts for your widgets, which the system inflates as appropriate when the screen orientation changes.

App widgets can now be displayed in third party launchers and other host apps through a new bind Intent (AppWidgetManager.ACTION\_APPWIDGET\_BIND).

### Simplified task navigation

Android 4.1 makes it easy for you to manage the “Up” navigation that’s available to users from inside of your apps and helps ensure a consistent experience for users.

You can **define the intended Up navigation** for individual Activity components of your UI by adding a new **XML attribute** in the app’s manifest file. At run time, as Activities are launched, the system extracts the Up navigation tree from the manifest file and automatically creates the Up affordance navigation in the action bar. Developers who declare Up navigation in the manifest no longer need to manage navigation by callback at run time, although they can also do so if needed.

Also available is a new **TaskStackBuilder** class that lets you quickly put together a synthetic task stack to start immediately or to use when an Activity is launched from a PendingIntent. Creating a synthetic task stack is especially useful when users launch Activities from remote views, such as from Home screen widgets and notifications, because it lets the developer provide a managed, consistent experience on Back navigation.

### Easy animations for Activity launch

You can use a new helper class, **ActivityOptions**, to create and control the animation displayed when you launch your Activities. Through the helper class, you can specify custom animation resources to be used when the activity is launched, or request new zoom animations that start from any rectangle you specify on screen and that optionally include a thumbnail bitmap.

### Transitions to Lights Out and Full Screen Modes

New system UI flags in View let you to cleanly transition from a normal application UI (with action bar, navigation bar, and system bar visible), to "lights out mode" (with status bar and action bar hidden and navigation bar dimmed) or "full screen mode" (with status bar, action bar, and navigation bar all hidden).

### New types of remoteable Views

Developers can now use **GridLayout** and **ViewStub** views in Home screen widgets and notifications. GridLayout lets you structure the content of your remote views and manage child views alignments with a shallower UI hierarchy. ViewStub is an invisible, zero-sized View that can be used to lazily inflate layout resources at runtime.

### Live wallpaper preview

Android 4.1 makes it easier for users to **find and install Live Wallpapers** from apps that include them. If your app includes Live Wallpapers, you can now start an Activity (ACTION\_CHANGE\_LIVE\_WALLPAPER) that shows the user a preview of the Live Wallpaper from your own app. From the preview, users can directly load the Live Wallpaper.

### Higher-resolution contact photos

With Android 4.1, you can store **contact photos** that are as large as **720 x 720**, making contacts even richer and more personal. Apps can store and retrieve contact photos at that size or use any other size needed. The maximum photo size supported on specific devices may vary, so apps should **query the built-in contacts provider** at run time to obtain the max size for the current device.

## New Input Types and Capabilities

### Find out about devices being added and removed

Apps can **register to be notified** when any new input devices are attached, by USB, Bluetooth, or any other connection type. They can use this information to change state or capabilities as needed. For example, a game could receive notification that a new keyboard or joystick is attached, indicating the presence of a new player.

### Query the capabilities of input devices

Android 4.1 includes APIs that let apps and games take full advantage of all input devices that are connected and available.

Apps can query the device manager to enumerate all of the input devices currently attached and learn about the capabilities of each.

### Control vibrator on input devices

Among other capabilities, apps can now make use of any **vibrator service** associated with an attached input device, such as for **Rumble Pak** controllers.

## Animation and Graphics

### Vsync for apps

Extending vsync across the Android framework leads to a more consistent framerate and a smooth, steady UI. So that apps also benefit, Android 4.1 **extends vsync timing** to all drawing and animations initiated by apps. This lets them optimize operations on the UI thread and provides a stable timebase for synchronization.

Apps can take advantage of vsync timing for free, through Android’s **animation framework**. The animation framework now uses vsync timing to automatically handle synchronization across animators.

For specialized uses, apps can access vsync timing through APIs exposed by a new Choreographer class. Apps can request invalidation on the next vsync frame — a good way to schedule animation when the app is not using the animation framework. For more advanced uses, apps can post a callback that the Choreographer class will run on the next frame.

### New animation actions and transition types

The animation framework now lets you define start and end actions to take when running ViewPropertyAnimator animations, to help synchronize them with other animations or actions in the application. The action can run any runnable object. For example, the runnable might specify another animation to start when the previous one finishes.

You can also now specify that a ViewPropertyAnimator use a layer during the course of its animation. Previously, it was a best practice to animate complicated views by setting up a layer prior to starting an animation and then handling an onAnimationEnd() event to remove the layer when the animation finishes. Now, the withLayer() method on ViewPropertyAnimator simplifies this process with a single method call.

A new transition type in LayoutTransition enables you to automate animations in response to all layout changes in a ViewGroup.

## New Types of Connectivity

### Android Beam

Android Beam is a popular NFC-based technology that lets users instantly share, just by touching two NFC-enabled phones together.

In Android 4.1, Android Beam makes it easier to share images, videos, or other payloads by **leveraging Bluetooth for the data transfer**. When the user triggers a transfer, Android Beam hands over from NFC to Bluetooth, making it really easy to manage the transfer of a file from one device to another.

### Wi-Fi Network Service Discovery

Android 4.1 introduces support for multicast **DNS-based service discovery**, which lets applications find and connect to services offered by peer devices over Wi-Fi networks — including mobile devices, printers, cameras, media players, and others. Developers can take advantage of Wi-Fi network service discovery to build cross-platform or multiplayer games and application experiences.

Using the service discovery API, apps can create and register any kind of service, for any other NSD-enabled device to discover. The service is advertised by multicast across the network using a human-readable string identifier, which lets user more easily identify the type of service.

Consumer devices can use the API to scan and discover services available from devices connected to the local Wi-Fi network. After discovery, apps can use the API to resolve the service to an IP address and port through which it can establish a socket connection.

You can take advantage of this API to build new features into your apps. For example, you could let users connect to a webcam, a printer, or an app on another mobile device that supports Wi-Fi peer-to-peer connections.

### Wi-Fi Direct Service Discovery

Ice Cream Sandwich introduced support for Wi-Fi Direct, a technology that lets apps **discover and pair directly**, over a high-bandwidth peer-to-peer connection. Wi-Fi Direct is an ideal way to share media, photos, files and other types of data and sessions, even where there is no cell network or Wi-Fi available.

Android 4.1 takes Wi-Fi Direct further, adding API support for **pre-associated service discovery**. Pre-associated service discovery lets your apps get more useful information from nearby devices about the services they support, before they attempt to connect. Apps can initiate discovery for a specific service and filter the list of discovered devices to those that actually support the target service or application.

For example, this means that your app could discover only devices that are “printers” or that have a specific game available, instead of discovering all nearby Wi-Fi Direct devices. On the other hand, your app can advertise the service it provides to other devices, which can discover it and then negotiate a connection. This greatly simplifies discovery and pairing for users and lets apps take advantage of Wi-Fi Direct more effectively.

With Wi-Fi Direct service discovery, you can create apps and **multiplayer games** that can share photos, videos, gameplay, scores, or almost anything else — all without requiring any Internet or mobile network. Your users can connect using only a direct p2p connection, which avoids using mobile bandwidth.

### Network Bandwidth Management

Android 4.1 helps apps **manage data usage** appropriately when the device is **connected to a metered network**, including tethering to a mobile hotspot. Apps can query whether the current network is metered before beginning a large download that might otherwise be relatively expensive to the user. Through the API, you can now get a clear picture of which networks are sensitive to data usage and manage your network activity accordingly.

## New Media Capabilities

### Media codec access

Android 4.1 provides low-level access to platform hardware and software codecs. Apps can query the system to discover what **low-level media codecs** are available on the device and then and use them in the ways they need. For example, you can now create multiple instances of a media codec, queue input buffers, and receive output buffers in return. In addition, the media codec framework supports protected content. Apps can query for an available codec that is able to play protected content with a DRM solution available on the device.

### USB Audio

USB audio output support allows hardware vendors to build hardware such as **audio docks** that interface with Android devices. This functionality is also exposed with the Android **Open Accessory Development Kit** (ADK) to give all developers the chance to create their own hardware.

### Audio record triggering

Android now lets you **trigger audio recording** based on the completion of an audio playback track. This is useful for situations such as playing back a tone to cue your users to begin speaking to record their voices. This feature helps you sync up recording so you don’t record audio that is currently being played back and prevents recordings from beginning too late.

### Multichannel audio

Android 4.1 supports **multichannel audio** on devices that have hardware multichannel audio out through the **HDMI port**. Multichannel audio lets you deliver rich media experiences to users for applications such as games, music apps, and video players. For devices that do not have the supported hardware, Android automatically downmixes the audio to the number of channels that are supported by the device (usually stereo).

Android 4.1 also adds built-in support for encoding/decoding AAC 5.1 audio.

### Audio preprocessing

Developers can apply **preprocessing effects** to audio being recorded, such as to apply noise suppression for improving speech recording quality, echo cancellation for acoustic echo, and auto gain control for audio with inconsistent volume levels. Apps that require high quality and clean audio recording will benefit from these preprocessors.

### Audio chaining

MediaPlayer supports **chaining audio streams together** to play audio files without pauses. This is useful for apps that require seamless transitions between audio files such as music players to play albums with continuous tracks or games.

### Media Router

The new APIs MediaRouter, MediaRouteActionProvider, and MediaRouteButton provide standard mechanisms and UI for**choosing where to play media**. Support is built-in for wired headsets and a2dp bluetooth headsets and speakers, and you can add your own routing options within your own app.

## Renderscript Computation

Android 4.1 extends Renderscript computation to give you more flexibility. You can now **sample textures** in your Renderscript compute scripts, and **new pragmas** are available to define the floating point precision required by your scripts. This lets you enable **NEON instructions** such as fast vector math operations on the CPU path, that wouldn’t otherwise be possible with the full IEEE 754-2008 standard.

You can now **debug** your Renderscript compute scripts on **x86-based emulator and hardware devices**. You can also define multiple root-style kernels in a single Renderscript source file.

## Android Browser and WebView

In Android 4.1, the Android Browser and WebViews include these enhancements:

* Better HTML5 video user experience, including touch-to-play/pause and smooth transition from inline to full screen mode.
* Improved rendering speed and reduced memory usage for better scrolling and zooming performance.
* Improved HTML5/CSS3/Canvas animation performance.
* Improved text input.
* Updated JavaScript Engine (V8) for better JavaScript performance.
* Support for the updated HTML5 Media Capture specification (the "capture" attribute on input type=file elements).

## Google APIs and services

To extend the capabilities of Android even further, several new services for Android are available.

### Google Cloud Messaging for Android

Google Cloud Messaging (GCM) is a service that lets developers send **short message data** to their users on Android devices, without needing a proprietary sync solution.

GCM handles all the details of **queuing messages and delivering them** efficiently to the targeted Android devices. It supports message **multicasting** and can reach up to 1000 connected devices simultaneously with a single request. It also supports message **payloads**, which means that in addition to sending tickle messages to an app on the device, developers can send up to 4K of data.

Google Cloud Messaging is completely **free for all developers** and sign-up is easy. See the Google Cloud Messaging page for registration, downloads, and documentation.

### App Encryption

Starting with Android 4.1, Google Play will help protect application assets by encrypting all paid apps with a device-specific key before they are delivered and stored on a device.

### Smart App Updates

Smart app updates is a new feature of Google Play that introduces a better way of delivering **app updates** to devices. When developers publish an update, Google Play now delivers only the **bits that have changed** to devices, rather than the entire APK. This makes the updates much lighter-weight in most cases, so they are faster to download, save the device’s battery, and conserve bandwidth usage on users’ mobile data plan. On average, a smart app update is about **1/3 the size**of a full APK update.

### Google Play services

Google Play services helps developers to **integrate Google services** such as authentication and Google+ into their apps delivered through Google Play.

Google Play services is automatically provisioned to end user devices by Google Play, so all you need is a **thin client library** in your apps.

Because your app only contains the small client library, you can take advantage of these services without a big increase in download size and storage footprint. Also, Google Play will **deliver regular updates** to the services, without developers needing to publish app updates to take advantage of them.

For more information about the APIs included in Google Play Services, see the Google Play services developer page.

|  |
| --- |
|  |
|  |
| Ice Cream Sandwich  Welcome to Android 4.0!  Android 4.0 delivers a refined, unified UI for phones and tablets and introduces innovative features for users and developers. This document provides a glimpse of the many new features and technologies that make Android 4.0 simple, beautiful, and beyond smart.   * Android 4.0 for Users * Android 4.0 for Developers   Android 4.0 for Users  http://developer.android.com/sdk/images/4.0/home.pnghttp://developer.android.com/sdk/images/4.0/lock.png  Simple, beautiful, beyond smart  Android 4.0 builds on the things people love most about Android — easy multitasking, rich notifications, customizable home screens, resizable widgets, and deep interactivity — and adds powerful new ways of communicating and sharing.  **Refined, evolved UI**  Focused on bringing the power of Android to the surface, Android 4.0 makes **common actions more visible** and lets users navigate with simple, intuitive gestures. Refined**animations** and feedback throughout the system make interactions engaging and interesting. An entirely **new typeface** optimized for high-resolution screens improves readability and brings a polished, modern feel to the user interface.  Virtual buttons in the System Bar let users navigate instantly to Back, Home, and Recent Apps. The **System Bar** and virtual buttons are present across all apps, but can be dimmed by applications for full-screen viewing. Users can access each application's contextual options in the **Action Bar**, displayed at the top (and sometimes also at the bottom) of the screen.  **Multitasking** is a key strength of Android and it's made even easier and more visual on Android 4.0. The Recent Apps button lets users jump instantly from one task to another using the list in the System Bar. The list pops up to show thumbnail images of apps used recently — tapping a thumbnail switches to the app.  http://developer.android.com/sdk/images/4.0/tasks.png  The Recent Apps list makes multitasking simple.  http://developer.android.com/sdk/images/4.0/lock-camera.png  Jump to the camera or see notifications without unlocking.  http://developer.android.com/sdk/images/4.0/contact-call.png  For incoming calls, you can respond instantly by text.  Rich and interactive **notifications** let users keep in constant touch with incoming messages, play music tracks, see real-time updates from apps, and much more. On smaller-screen devices, notifications appear at the top of the screen, while on larger-screen devices they appear in the System Bar.  http://developer.android.com/sdk/images/4.0/allapps.png http://developer.android.com/sdk/images/4.0/calendar-widget.png  The All Apps launcher (left) and resizable widgets (right) give you apps and rich content from the home screen.  **Home screen folders and favorites tray**  New home screen **folders** offer a new way for users to group their apps and shortcuts logically, just by dragging one onto another. Also, in All Apps launcher, users can now simply **drag an app** to get information about it or immediately uninstall it, or disable a pre-installed app.  On smaller-screen devices, the home screen now includes a customizable **favorites tray** visible from all home screens. Users can drag apps, shortcuts, folders, and other priority items in or out of the favorites tray for instant access from any home screen.  **Resizable widgets**  Home screens in Android 4.0 are designed to be content-rich and customizable. Users can do much more than add shortcuts — they can embed live application content directly through interactive **widgets**. Widgets let users check email, flip through a calendar, play music, check social streams, and more — right from the home screen, without having to launch apps. Widgets are resizable, so users can expand them to show more content or shrink them to save space.  **New lock screen actions**  The lock screens now let users do more without unlocking. From the slide lock screen, users can **jump directly to the camera** for a picture or **pull down the notifications window** to check for messages. When listening to music, users can even manage music tracks and see album art.  **Quick responses for incoming calls**  When an incoming call arrives, users can now quickly **respond by text message**, without needing to pick up the call or unlock the device. On the incoming call screen, users simply slide a control to see a list of text responses and then tap to send and end the call. Users can add their own responses and manage the list from the Settings app.  **Swipe to dismiss notifications, tasks, and browser tabs**  Android 4.0 makes managing notifications, recent apps, and browser tabs even easier. Users can now dismiss individual notifications, apps from the Recent Apps list, and browser tabs with a simple swipe of a finger.  http://developer.android.com/sdk/images/4.0/text-replace.png  A spell-checker lets you find errors and fix them faster.  http://developer.android.com/sdk/images/4.0/tts.png  A powerful voice input engine lets you dictate continously.  **Improved text input and spell-checking**  The soft keyboard in Android 4.0 makes text input even faster and more accurate. Error correction and word suggestion are improved through a new set of default dictionaries and more accurate heuristics for handling cases such as double-typed characters, skipped letters, and omitted spaces. Word suggestion is also improved and the suggestion strip is simplified to show only three words at a time.  To fix misspelled words more easily, Android 4.0 adds a **spell-checker** that locates and underlines errors and suggests replacement words. With one tap, users can choose from multiple spelling suggestions, delete a word, or add it to the dictionary. Users can even tap to see replacement suggestions for words that are spelled correctly. For specialized features or additional languages, users can now download and install third-party dictionaries, spell-checkers, and other text services.  **Powerful voice input engine**  Android 4.0 introduces a powerful new voice input engine that offers a continuous "open microphone" experience and streaming voice recognition. The new voice input engine lets users dictate the text they want, for as long as they want, using the language they want. Users can **speak continously** for a prolonged time, even pausing for intervals if needed, and dictate punctuation to create correct sentences. As the voice input engine enters text, it underlines possible dictation errors in gray. After dictating, users can tap the underlined words to quickly replace them from a list of suggestions.  http://developer.android.com/sdk/images/4.0/usage-all.png http://developer.android.com/sdk/images/4.0/usage-maps.png  Data usage controls let you monitor total usage by network type and application and then set limits if needed.  **Control over network data**  Mobile devices can make extensive use of network data for streaming content, synchronizing data, downloading apps, and more. To meet the needs of users with**tiered or metered data plans**, Android 4.0 adds new controls for managing network data usage.  In the Settings app, colorful charts show the total data usage on each network type (mobile or Wi-Fi), as well as amount of data used by each running application. Based on their data plans, users can optionally set warning levels or hard limits on data usage or disable mobile data altogether. Users can also manage the background data used by individual applications as needed.  **Designed for accessibility**  A variety of new features greatly enhance the accessibility of Android 4.0 for blind or visually impaired users. Most important is a new **explore-by-touch mode** that lets users navigate without having to see the screen. Touching the screen once triggers audible feedback that identifies the UI component below; a second touch in the same component activates it with a full touch event. The new mode is especially important to support users on new devices that use virtual buttons in the System Bar, rather than dedicated hardware buttons or trackballs. Also, standard apps are updated to offer an improved accessibility experience. The **Browser** supports a script-based screen reader for reading favorite web content and navigating sites. For improved readability, users can also increase the default font size used across the system.  The accessibility experience begins at first setup — a simple **touch gesture** during setup (clockwise square from upper left) activates all accessibility features and loads a setup tutorial. Once accessibility features are active, everything visible on the screen can be spoken aloud by the standard screen reader.  Communication and sharing  http://developer.android.com/sdk/images/4.0/contact-faves.png http://developer.android.com/sdk/images/4.0/contact-connect.png http://developer.android.com/sdk/images/4.0/contact-email.png  Contacts and profiles are integrated across apps and social networks, for a consistent, personal experience everywhere — from incoming calls to emails.  Designed for the way people live, Android 4.0 integrates rich social communication and sharing touchpoints across the system, making it easy to talk, email, text, and share.  **People and profiles**  Throughout the system, a user’s social groups, profiles, and contacts are linked together and integrated for easy accessibility. At the center is a new**People app** that offers richer profile information, including a large profile picture, phone numbers, addresses and accounts, status updates, events, stream items, and a new button for connecting on integrated social networks.  The user's own contact information is stored in a new **"Me" profile**, allowing easier sharing with apps and people. All of the user's integrated contacts are displayed in an easy to manage list, including controls over which contacts are shown from any integrated account or social network. Wherever the user navigates across the system, tapping a profile photo displays Quick Contacts, with large profile pictures, shortcuts to phone numbers, text messaging, and more.  **Unified calendar, visual voicemail**  To help organize appointments and events, an updated **Calendar app** brings together personal, work, school, and social agendas. With user permission, other applications can contribute events to the calendar and manage reminders, for an integrated view across multiple calendar providers. The app is redesigned to let users manage events more easily. Calendars are color-coded and users can **swipe left or right** to change dates and pinch to zoom in or out agendas.  In the phone app, a new **visual voicemail** features integrates incoming messages, voice transcriptions, and audio files from one or more providers. Third-party applications can integrate with the Phone app to add their own voice messages, transcriptions, and more to the visual voicemail inbox.  http://developer.android.com/sdk/images/4.0/camera.pnghttp://developer.android.com/sdk/images/4.0/gallery-edit.pnghttp://developer.android.com/sdk/images/4.0/gallery-share.png  Capture the picture you want, edit, and share instantly.  **Rich and versatile camera capabilities**  The Camera app includes many new features that let users capture special moments with great photos and videos. After capturing images, they can edit and share them easily with friends.  When taking pictures, **continuous focus**, **zero shutter lag exposure**, and decreased shot-to-shot speed help capture clear, precise images. **Stabilized image zoom** lets users compose photos and video in the way they want, including while video is recording. For new flexibility and convenience while shooting video, users can now take **snapshots at full video resolution** just by tapping the screen as video continues to record.  To make it easier to take great pictures of people, built-in **face detection**locates faces in the frame and automatically sets focus. For more control, users can **tap to focus** anywhere in the preview image.  For capturing larger scenes, the Camera introduces a **single-motion panorama** mode. In this mode, the user starts an exposure and then slowly turns the Camera to encompass as wide a perspective as needed. The Camera assembles the full range of continuous imagery into a single panoramic photo.  After taking a picture or video, users can quickly share it by email, text message, bluetooth, social networks, and more, just by tapping the thumbnail in the camera controls.  http://developer.android.com/sdk/images/4.0/gallery-widget.png  A Photo Gallery widget on the home screen.  **Redesigned Gallery app with photo editor**  The Gallery app now makes it easier to manage, show, and share photos and videos. For managing collections, a **redesigned album layout** shows many more albums and offers larger thumbnails. There are many ways to sort albums, including by time, location, people, and tags. To help pictures look their best, the Gallery now includes a powerful **photo editor**. Users can crop and rotate pictures, set levels, remove red eyes, add effects, and much more. After retouching, users can select one or multiple pictures or videos to share instantly over email, text messaging, bluetooth, social networks, or other apps.  An improved **Picture Gallery widget** lets users look at pictures directly on their home screen. The widget can display pictures from a selected album, shuffle pictures from all albums, or show a single image. After adding the widget to the home screen, users can flick through the photo stacks to locate the image they want, then tap to load it in Gallery.  http://developer.android.com/sdk/images/4.0/live-effects.png  Live Effects let you change backgrounds and use Silly Faces during video.  **Live Effects for transforming video**  Live Effects is a collection of graphical transformations that add interest and fun to videos captured in the Camera app. For example, users can **change the background** behind them to any stock or custom image, for just the right setting when shooting videeo. Also available for video is Silly Faces, a set of morphing effects that use state-of-the-art face recognition and GPU filters to transform facial features. For example, you can use effects such as small eyes, big mouth, big nose, face squeeze, and more. Outside of the Camera app, Live Effects is available during video chat in the Google Talk app.  http://developer.android.com/sdk/images/4.0/screenshot.png  Snapping a screenshot.  **Sharing with screenshots**  Users can now share what's on their screens more easily by taking screenshots. Hardware buttons let them snap a **screenshot** and store it locally. Afterward, they can view, edit, and share the screen shot in Gallery or a similar app.  Cloud-connected experience  http://developer.android.com/sdk/images/4.0/browser-tabs.png http://developer.android.com/sdk/images/4.0/browser.png  The Browser tabs menu (left) lets you quickly switch browser tabs. The options menu (right) gives you new ways to manage your browsing experience.  http://developer.android.com/sdk/images/4.0/bbench.png  Benchmark comparisons of Android Browser.  Android has always been cloud-connected, letting users browse the web and sync photos, apps, games, email, and contacts — wherever they are and across all of their devices. Android 4.0 adds new browsing and email capabilities to let users take even more with them and keep communication organized.  **Powerful web browsing**  The Android Browser offers an experience that’s as rich and convenient as a desktop browser. It lets users instantly sync and manage **Google Chrome bookmarks** from all of their accounts, jump to their favorite content faster, and even save it for reading later in case there's no network available.  To get the most out of web content, users can now request full**desktop versions** of web sites, rather than their mobile versions. Users can set their preference for web sites separately for each**browser tab**. For longer content, users can save a copy for **offline reading**. To find and open saved pages, users can browse a visual list that’s included with browser bookmarks and history. For better readability and accessibility, users can increase the browser’s **zoom levels** and override the system default **text sizes**.  Across all types of content, the Android Browser offers dramatically improved **page rendering performance** through updated versions of the WebKit core and the V8 Crankshaft compilation engine for JavaScript. In benchmarks run on a Nexus S device, the Android 4.0 browser showed an improvement of nearly 220% over the Android 2.3 browser in the V8 Benchmark Suite and more than 35% in the SunSpider 9.1 JavaScript Benchmark. When run on a Galaxy Nexus device, the Android 4.0 browser showed improvement of nearly 550% in the V8 benchmark and nearly 70% in the SunSpider benchmark.  **Improved email**  In Android 4.0, email is easier to send, read, and manage. For composing email, **improved auto-completion** of recipients helps with finding and adding frequent contacts more quickly. For easier input of frequent text, users can now create**quick responses** and store them in the app, then enter them from a convenient menu when composing. When replying to a message, users can now toggle the message to Reply All and Forward without changing screens.  For easier browsing across accounts and labels, the app adds an **integrated menu** of accounts and recent labels. To help users locate and organize IMAP and Exchange email, the Email app now supports **nested mail subfolders**, each with synchronization rules. Users can also search across folders on the server, for faster results.  For **enterprises**, the Email app supports EAS v14. It supports EAS certificate authentication, provides ABQ strings for device type and mode, and allows automatic sync to be disabled while roaming. Administrators can also limit attachment size or disable attachments.  For keeping track of incoming email more easily, a **resizable Email widget** lets users flick through recent email right from the home screen, then jump into the Email app to compose or reply.  http://developer.android.com/sdk/images/4.0/beam.png  Android Beam lets users share what they are using with a single tap.  Innovation  Android is continously driving innovation forward, pushing the boundaries of communication and sharing with new capabilities and interactions.  **Android Beam for NFC-based sharing**  Android Beam is an innovative, convenient feature for sharing across two NFC-enabled devices, It lets people instantly exchange favorite apps, contacts, music, videos — almost anything. It’s  Android 4.0.3 APIs API Level: **15**  Android 4.0.3 (ICE\_CREAM\_SANDWICH\_MR1) is an incremental release of the Android 4.x (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. API Overview The sections below provide a technical overview of new APIs in Android 4.0.3.  http://developer.android.com/assets/images/triangle-opened.png**Table of Contents**   1. Social stream API in Contacts Provider 2. Calendar Provider 3. Home screen widgets 4. Spell-checking 5. Bluetooth 6. UI toolkit 7. Accessibility 8. Text-to-speech 9. Database 10. Intents 11. Camera 12. Permissions   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 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 byContactsContract.StreamItemPhotos, which is available as a sub-directory of theContactsContract.StreamItems Uri.  See ContactsContract.StreamItems and ContactsContract.StreamItemPhotos for more information.  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. 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 asEVENT\_COLOR\_KEY for events.  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. 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.  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. 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. 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.  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.  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.  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 CrossProcessCursor objects 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.  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. 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 theQUALITY\_QVGA.and QUALITY\_TIME\_LAPSE\_QVGA constants. * New methods setVideoStabilization(), setVideoStabilization(), andisVideoStabilizationSupported() let you check and manage video stabilization for a Camera.  Permissions The following are new permissions:   * READ\_SOCIAL\_STREAM and WRITE\_SOCIAL\_STREAM: Allow a sync adapter to read and write social stream data to a contact in the shared Contacts Provider.   For a detailed view of all API changes in Android 4.0.3 (API Level 15), see the API Differences Report. 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 anandroid:minSdkVersion="15" attribute to the <uses-sdk> element.  For more information, see the API Levels document.   |  |  | | --- | --- | | Android 4.0 APIs API Level: **14**  Android 4.0 (ICE\_CREAM\_SANDWICH) is a major platform release that adds a variety of new features for users and app developers. Besides all the new features and APIs discussed below, Android 4.0 is an important platform release because it brings the extensive set of APIs and Holographic themes from Android 3.x to smaller screens. As an app developer, you now have a single platform and unified API framework that enables you to develop and publish your application with a single APK that provides an optimized user experience for handsets, tablets, and more, when running the same version of Android—Android 4.0 (API level 14) or greater.  For developers, the Android 4.0 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, use the Android SDK Manager to download the platform into your SDK. API Overview The sections below provide a technical overview of new APIs in Android 4.0.  http://developer.android.com/assets/images/triangle-opened.png **Table of Contents**   1. Social APIs in Contacts Provider 2. Calendar Provider 3. Voicemail Provider 4. Multimedia 5. Camera 6. Android Beam (NDEF Push with NFC) 7. Wi-Fi P2P 8. Bluetooth Health Devices 9. Accessibility 10. Spell Checker Services 11. Text-to-speech Engines 12. Network Usage 13. RenderScript 14. Enterprise 15. Device Sensors 16. Action Bar 17. User Interface and Views 18. Input Framework 19. Properties 20. Hardware Acceleration 21. JNI Changes 22. WebKit 23. Permissions 24. Device Features  Social APIs in Contacts Provider The contact APIs defined by the ContactsContract provider have been extended to support new social-oriented features such as a personal profile for the device owner and the ability for users to invite individual contacts to social networks that are installed on the device. User Profile Android now includes a personal profile that represents the device owner, as defined by theContactsContract.Profile table. Social apps that maintain a user identity can contribute to the user's profile data by creating a new ContactsContract.RawContacts entry within the ContactsContract.Profile. That is, raw contacts that represent the device user do not belong in the traditional raw contacts table defined by theContactsContract.RawContacts Uri; instead, you must add a profile raw contact in the table atCONTENT\_RAW\_CONTACTS\_URI. Raw contacts in this table are then aggregated into the single user-visible profile labeled "Me".  Adding a new raw contact for the profile requires the WRITE\_PROFILE permission. Likewise, in order to read from the profile table, you must request the READ\_PROFILE permission. However, most apps should not need to read the user profile, even when contributing data to the profile. Reading the user profile is a sensitive permission and you should expect users to be skeptical of apps that request it. Invite Intent The INVITE\_CONTACT intent action allows an app to invoke an action that indicates the user wants to add a contact to a social network. The app receiving the app uses it to invite the specified contact to that social network. Most apps will be on the receiving-end of this operation. For example, the built-in People app invokes the invite intent when the user selects "Add connection" for a specific social app that's listed in a person's contact details.  To make your app visible as in the "Add connection" list, your app must provide a sync adapter to sync contact information from your social network. You must then indicate to the system that your app responds to theINVITE\_CONTACT intent by adding the inviteContactActivity attribute to your app’s sync configuration file, with a fully-qualified name of the activity that the system should start when sending the invite intent. The activity that starts can then retrieve the URI for the contact in question from the intent’s data and perform the necessary work to invite that contact to the network or add the person to the user’s connections.  See the Sample Sync Adapter app for an example (specifically, see the contacts.xml file). Large photos Android now supports high resolution photos for contacts. Now, when you push a photo into a contact record, the system processes it into both a 96x96 thumbnail (as it has previously) and a 256x256 "display photo" that's stored in a new file-based photo store (the exact dimensions that the system chooses may vary in the future). You can add a large photo to a contact by putting a large photo in the usual PHOTO column of a data row, which the system will then process into the appropriate thumbnail and display photo records. Contact Usage Feedback The new ContactsContract.DataUsageFeedback APIs allow you to help track how often the user uses particular methods of contacting people, such as how often the user uses each phone number or e-mail address. This information helps improve the ranking for each contact method associated with each person and provide better suggestions for contacting each person. Calendar Provider The new calendar APIs allow you to read, add, modify and delete calendars, events, attendees, reminders and alerts, which are stored in the Calendar Provider.  A variety of apps and widgets can use these APIs to read and modify calendar events. However, some of the most compelling use cases are sync adapters that synchronize the user's calendar from other calendar services with the Calendar Provider, in order to offer a unified location for all the user's events. Google Calendar events, for example, are synchronized with the Calendar Provider by the Google Calendar Sync Adapter, allowing these events to be viewed with Android's built-in Calendar app.  The data model for calendars and event-related information in the Calendar Provider is defined by CalendarContract. All the user’s calendar data is stored in a number of tables defined by various subclasses of CalendarContract:   * The CalendarContract.Calendars table holds the calendar-specific information. Each row in this table contains the details for a single calendar, such as the name, color, sync information, and so on. * The CalendarContract.Events table holds event-specific information. Each row in this table contains the information for a single event, such as the event title, location, start time, end time, and so on. The event can occur one time or recur multiple times. Attendees, reminders, and extended properties are stored in separate tables and use the event’s \_ID to link them with the event. * The CalendarContract.Instances table holds the start and end time for occurrences of an event. Each row in this table represents a single occurrence. For one-time events there is a one-to-one mapping of instances to events. For recurring events, multiple rows are automatically generated to correspond to the multiple occurrences of that event. * The CalendarContract.Attendees table holds the event attendee or guest information. Each row represents a single guest of an event. It specifies the type of guest the person is and the person’s response for the event. * The CalendarContract.Reminders table holds the alert/notification data. Each row represents a single alert for an event. An event can have multiple reminders. The number of reminders per event is specified in MAX\_REMINDERS, which is set by the sync adapter that owns the given calendar. Reminders are specified in number-of-minutes before the event is scheduled and specify an alarm method such as to use an alert, email, or SMS to remind the user. * The CalendarContract.ExtendedProperties table hold opaque data fields used by the sync adapter. The provider takes no action with items in this table except to delete them when their related events are deleted.   To access a user’s calendar data with the Calendar Provider, your application must request the READ\_CALENDARpermission (for read access) and WRITE\_CALENDAR (for write access). Event intent If all you want to do is add an event to the user’s calendar, you can use an ACTION\_INSERT intent with the data defined byEvents.CONTENT\_URI in order to start an activity in the Calendar app that creates new events. Using the intent does not require any permission and you can specify event details with the following extras:   * Events.TITLE: Name for the event * CalendarContract.EXTRA\_EVENT\_BEGIN\_TIME: Event begin time in milliseconds from the epoch * CalendarContract.EXTRA\_EVENT\_END\_TIME: Event end time in milliseconds from the epoch * Events.EVENT\_LOCATION: Location of the event * Events.DESCRIPTION: Event description * Intent.EXTRA\_EMAIL: Email addresses of those to invite * Events.RRULE: The recurrence rule for the event * Events.ACCESS\_LEVEL: Whether the event is private or public * Events.AVAILABILITY: Whether the time period of this event allows for other events to be scheduled at the same time  Voicemail Provider The new Voicemail Provider allows applications to add voicemails to the device, in order to present all the user's voicemails in a single visual presentation. For instance, it’s possible that a user has multiple voicemail sources, such as one from the phone’s service provider and others from VoIP or other alternative voice services. These apps can use the Voicemail Provider APIs to add their voicemails to the device. The built-in Phone application then presents all voicemails to the user in a unified presentation. Although the system’s Phone application is the only application that can read all the voicemails, each application that provides voicemails can read those that it has added to the system (but cannot read voicemails from other services).  Because the APIs currently do not allow third-party apps to read all the voicemails from the system, the only third-party apps that should use the voicemail APIs are those that have voicemail to deliver to the user.  The VoicemailContract class defines the content provider for the Voicemail Provder. The subclassesVoicemailContract.Voicemails and VoicemailContract.Status provide tables in which apps can insert voicemail data for storage on the device. For an example of a voicemail provider app, see the Voicemail Provider Demo. Multimedia Android 4.0 adds several new APIs for applications that interact with media such as photos, videos, and music. Media Effects A new media effects framework allows you to apply a variety of visual effects to images and videos. For example, image effects allow you to easily fix red-eye, convert an image to grayscale, adjust brightness, adjust saturation, rotate an image, apply a fisheye effect, and much more. The system performs all effects processing on the GPU to obtain maximum performance.  For maximum performance, effects are applied directly to OpenGL textures, so your application must have a valid OpenGL context before it can use the effects APIs. The textures to which you apply effects may be from bitmaps, videos or even the camera. However, there are certain restrictions that textures must meet:   1. They must be bound to a GL\_TEXTURE\_2D texture image 2. They must contain at least one mipmap level   An Effect object defines a single media effect that you can apply to an image frame. The basic workflow to create anEffect is:   1. Call EffectContext.createWithCurrentGlContext() from your OpenGL ES 2.0 context. 2. Use the returned EffectContext to call EffectContext.getFactory(), which returns an instance ofEffectFactory. 3. Call createEffect(), passing it an effect name from @link android.media.effect.EffectFactory}, such asEFFECT\_FISHEYE or EFFECT\_VIGNETTE.   You can adjust an effect’s parameters by calling setParameter() and passing a parameter name and parameter value. Each type of effect accepts different parameters, which are documented with the effect name. For example,EFFECT\_FISHEYE has one parameter for the scale of the distortion.  To apply an effect on a texture, call apply() on the Effect and pass in the input texture, it’s width and height, and the output texture. The input texture must be bound to a GL\_TEXTURE\_2D texture image (usually done by calling theglTexImage2D() function). You may provide multiple mipmap levels. If the output texture has not been bound to a texture image, it will be automatically bound by the effect as a GL\_TEXTURE\_2D and with one mipmap level (0), which will have the same size as the input.  All effects listed in EffectFactory are guaranteed to be supported. However, some additional effects available from external libraries are not supported by all devices, so you must first check if the desired effect from the external library is supported by calling isEffectSupported(). Remote control client The new RemoteControlClient allows media players to enable playback controls from remote control clients such as the device lock screen. Media players can also expose information about the media currently playing for display on the remote control, such as track information and album art.  To enable remote control clients for your media player, instantiate a RemoteControlClient with its constructor, passing it a PendingIntent that broadcasts ACTION\_MEDIA\_BUTTON. The intent must also declare the explicitBroadcastReceiver component in your app that handles the ACTION\_MEDIA\_BUTTON event.  To declare which media control inputs your player can handle, you must call setTransportControlFlags() on yourRemoteControlClient, passing a set of FLAG\_KEY\_MEDIA\_\* flags, such as FLAG\_KEY\_MEDIA\_PREVIOUS andFLAG\_KEY\_MEDIA\_NEXT.  You must then register your RemoteControlClient by passing it toMediaManager.registerRemoteControlClient(). Once registered, the broadcast receiver you declared when you instantiated the RemoteControlClient will receive ACTION\_MEDIA\_BUTTON events when a button is pressed from a remote control. The intent you receive includes the KeyEvent for the media key pressed, which you can retrieve from the intent with getParcelableExtra(Intent.EXTRA\_KEY\_EVENT).  To display information on the remote control about the media playing, call editMetaData() and add metadata to the returned RemoteControlClient.MetadataEditor. You can supply a bitmap for media artwork, numerical information such as elapsed time, and text information such as the track title. For information on available keys see theMETADATA\_KEY\_\* flags in MediaMetadataRetriever.  For a sample implementation, see the Random Music Player, which provides compatibility logic such that it enables the remote control client on Android 4.0 devices while continuing to support devices back to Android 2.1. Media player  * Streaming online media from MediaPlayer now requires the INTERNET permission. If you use MediaPlayer to play content from the Internet, be sure to add the INTERNET permission to your manifest or else your media playback will not work beginning with Android 4.0. * setSurface() allows you define a Surface to behave as the video sink. * setDataSource() allows you to send additional HTTP headers with your request, which can be useful for HTTP(S) live streaming * HTTP(S) live streaming now respects HTTP cookies across requests  Media types Android 4.0 adds support for:   * HTTP/HTTPS live streaming protocol version 3 * ADTS raw AAC audio encoding * WEBP images * Matroska video   For more info, see Supported Media Formats. Camera The Camera class now includes APIs for detecting faces and controlling focus and metering areas. Face detection Camera apps can now enhance their abilities with Android’s face detection APIs, which not only detect the face of a subject, but also specific facial features, such as the eyes and mouth.  To detect faces in your camera application, you must register a Camera.FaceDetectionListener by callingsetFaceDetectionListener(). You can then start your camera surface and start detecting faces by callingstartFaceDetection().  When the system detects one or more faces in the camera scene, it calls the onFaceDetection() callback in your implementation of Camera.FaceDetectionListener, including an array of Camera.Face objects.  An instance of the Camera.Face class provides various information about the face detected, including:   * A Rect that specifies the bounds of the face, relative to the camera's current field of view * An integer betwen 1 and 100 that indicates how confident the system is that the object is a human face * A unique ID so you can track multiple faces * Several Point objects that indicate where the eyes and mouth are located   **Note:** Face detection may not be supported on some devices, so you should check by callinggetMaxNumDetectedFaces() and ensure the return value is greater than zero. Also, some devices may not support identification of eyes and mouth, in which case, those fields in the Camera.Face object will be null. Focus and metering areas Camera apps can now control the areas that the camera uses for focus and for metering white balance and auto-exposure. Both features use the new Camera.Area class to specify the region of the camera’s current view that should be focused or metered. An instance of the Camera.Area class defines the bounds of the area with a Rect and the area's weight—representing the level of importance of that area, relative to other areas in consideration—with an integer.  Before setting either a focus area or metering area, you should first call getMaxNumFocusAreas() orgetMaxNumMeteringAreas(), respectively. If these return zero, then the device does not support the corresponding feature.  To specify the focus or metering areas to use, simply call setFocusAreas() or setMeteringAreas(). Each take a Listof Camera.Area objects that indicate the areas to consider for focus or metering. For example, you might implement a feature that allows the user to set the focus area by touching an area of the preview, which you then translate to anCamera.Area object and request that the camera focus on that area of the scene. The focus or exposure in that area will continually update as the scene in the area changes. Continuous auto focus for photos You can now enable continuous auto focusing (CAF) when taking photos. To enable CAF in your camera app, passFOCUS\_MODE\_CONTINUOUS\_PICTURE to setFocusMode(). When ready to capture a photo, call autoFocus(). YourCamera.AutoFocusCallback immediately receives a callback to indicate whether focus was achieved. To resume CAF after receiving the callback, you must call cancelAutoFocus().  **Note:** Continuous auto focus is also supported when capturing video, using FOCUS\_MODE\_CONTINUOUS\_VIDEO, which was added in API level 9. Other camera features  * While recording video, you can now call takePicture() to save a photo without interrupting the video session. Before doing so, you should call isVideoSnapshotSupported() to be sure the hardware supports it. * You can now lock auto exposure and white balance with setAutoExposureLock() andsetAutoWhiteBalanceLock() to prevent these properties from changing. * You can now call setDisplayOrientation() while the camera preview is running. Previously, you could call this only before beginning the preview, but you can now change the orientation at any time.  Camera broadcast intents  * Camera.ACTION\_NEW\_PICTURE: This indicates that the user has captured a new photo. The built-in Camera app invokes this broadcast after a photo is captured and third-party camera apps should also broadcast this intent after capturing a photo. * Camera.ACTION\_NEW\_VIDEO: This indicates that the user has captured a new video. The built-in Camera app invokes this broadcast after a video is recorded and third-party camera apps should also broadcast this intent after capturing a video.  Android Beam (NDEF Push with NFC) Android Beam is a new NFC feature that allows you to send NDEF messages from one device to another (a process also known as “NDEF Push"). The data transfer is initiated when two Android-powered devices that support Android Beam are in close proximity (about 4 cm), usually with their backs touching. The data inside the NDEF message can contain any data that you wish to share between devices. For example, the People app shares contacts, YouTube shares videos, and Browser shares URLs using Android Beam.  To transmit data between devices using Android Beam, you need to create an NdefMessage that contains the information you want to share while your activity is in the foreground. You must then pass the NdefMessage to the system in one of two ways:   * Define a single NdefMessage to push while in the activity:   Call setNdefPushMessage() at any time to set the message you want to send. For instance, you might call this method and pass it your NdefMessage during your activity’s onCreate() method. Then, whenever Android Beam is activated with another device while the activity is in the foreground, the system sends the NdefMessage to the other device.   * Define the NdefMessage to push at the time that Android Beam is initiated:   Implement NfcAdapter.CreateNdefMessageCallback, in which your implementation of thecreateNdefMessage() method returns the NdefMessage you want to send. Then pass theNfcAdapter.CreateNdefMessageCallback implementation to setNdefPushMessageCallback().  In this case, when Android Beam is activated with another device while your activity is in the foreground, the system calls createNdefMessage() to retrieve the NdefMessage you want to send. This allows you to define theNdefMessage to deliver only once Android Beam is initiated, in case the contents of the message might vary throughout the life of the activity.  In case you want to run some specific code once the system has successfully delivered your NDEF message to the other device, you can implement NfcAdapter.OnNdefPushCompleteCallback and set it withsetNdefPushCompleteCallback(). The system will then call onNdefPushComplete() when the message is delivered.  On the receiving device, the system dispatches NDEF Push messages in a similar way to regular NFC tags. The system invokes an intent with the ACTION\_NDEF\_DISCOVERED action to start an activity, with either a URL or a MIME type set according to the first NdefRecord in the NdefMessage. For the activity you want to respond, you can declare intent filters for the URLs or MIME types your app cares about. For more information about Tag Dispatch see the NFC developer guide.  If you want your NdefMessage to carry a URI, you can now use the convenience method createUri to construct a newNdefRecord based on either a string or a Uri object. If the URI is a special format that you want your application to also receive during an Android Beam event, you should create an intent filter for your activity using the same URI scheme in order to receive the incoming NDEF message.  You should also pass an “Android application record" with your NdefMessage in order to guarantee that your application handles the incoming NDEF message, even if other applications filter for the same intent action. You can create an Android application record by calling createApplicationRecord(), passing it your application’s package name. When the other device receives the NDEF message with the application record and multiple applications contain activities that handle the specified intent, the system always delivers the message to the activity in your application (based on the matching application record). If the target device does not currently have your application installed, the system uses the Android application record to launch Google Play and take the user to the application in order to install it.  If your application doesn’t use NFC APIs to perform NDEF Push messaging, then Android provides a default behavior: When your application is in the foreground on one device and Android Beam is invoked with another Android-powered device, then the other device receives an NDEF message with an Android application record that identifies your application. If the receiving device has the application installed, the system launches it; if it’s not installed, Google Play opens and takes the user to your application in order to install it.  You can read more about Android Beam and other NFC features in the NFC Basics developer guide. For some example code using Android Beam, see the Android Beam Demo. Wi-Fi P2P Android now supports Wi-Fi peer-to-peer (P2P) connections between Android-powered devices and other device types (in compliance with the Wi-Fi Alliance's Wi-Fi Direct™ certification program) without a hotspot or Internet connection. The Android framework provides a set of Wi-Fi P2P APIs that allow you to discover and connect to other devices when each device supports Wi-Fi P2P, then communicate over a speedy connection across distances much longer than a Bluetooth connection.  A new package, android.net.wifi.p2p, contains all the APIs for performing peer-to-peer connections with Wi-Fi. The primary class you need to work with is WifiP2pManager, which you can acquire by callinggetSystemService(WIFI\_P2P\_SERVICE). The WifiP2pManager includes APIs that allow you to:   * Initialize your application for P2P connections by calling initialize() * Discover nearby devices by calling discoverPeers() * Start a P2P connection by calling connect() * And more   Several other interfaces and classes are necessary as well, such as:   * The WifiP2pManager.ActionListener interface allows you to receive callbacks when an operation such as discovering peers or connecting to them succeeds or fails. * WifiP2pManager.PeerListListener interface allows you to receive information about discovered peers. The callback provides a WifiP2pDeviceList, from which you can retrieve a WifiP2pDevice object for each device within range and get information such as the device name, address, device type, the WPS configurations the device supports, and more. * The WifiP2pManager.GroupInfoListener interface allows you to receive information about a P2P group. The callback provides a WifiP2pGroup object, which provides group information such as the owner, the network name, and passphrase. * WifiP2pManager.ConnectionInfoListener interface allows you to receive information about the current connection. The callback provides a WifiP2pInfo object, which has information such as whether a group has been formed and who is the group owner.   In order to use the Wi-Fi P2P APIs, your app must request the following user permissions:   * ACCESS\_WIFI\_STATE * CHANGE\_WIFI\_STATE * INTERNET (although your app doesn’t technically connect to the Internet, communicating to Wi-Fi P2P peers with standard java sockets requires Internet permission).   The Android system also broadcasts several different actions during certain Wi-Fi P2P events:   * WIFI\_P2P\_CONNECTION\_CHANGED\_ACTION: The P2P connection state has changed. This carriesEXTRA\_WIFI\_P2P\_INFO with a WifiP2pInfo object and EXTRA\_NETWORK\_INFO with a NetworkInfo object. * WIFI\_P2P\_STATE\_CHANGED\_ACTION: The P2P state has changed between enabled and disabled. It carriesEXTRA\_WIFI\_STATE with either WIFI\_P2P\_STATE\_DISABLED or WIFI\_P2P\_STATE\_ENABLED * WIFI\_P2P\_PEERS\_CHANGED\_ACTION: The list of peer devices has changed. * WIFI\_P2P\_THIS\_DEVICE\_CHANGED\_ACTION: The details for this device have changed.   See the WifiP2pManager documentation for more information. Also look at the Wi-Fi P2P Demo sample application. Bluetooth Health Devices Android now supports Bluetooth Health Profile devices, so you can create applications that use Bluetooth to communicate with health devices that support Bluetooth, such as heart-rate monitors, blood meters, thermometers, and scales.  Similar to regular headset and A2DP profile devices, you must call getProfileProxy() with aBluetoothProfile.ServiceListener and the HEALTH profile type to establish a connection with the profile proxy object.  Once you’ve acquired the Health Profile proxy (the BluetoothHealth object), connecting to and communicating with paired health devices involves the following new Bluetooth classes:   * BluetoothHealthCallback: You must extend this class and implement the callback methods to receive updates about changes in the application’s registration state and Bluetooth channel state. * BluetoothHealthAppConfiguration: During callbacks to your BluetoothHealthCallback, you’ll receive an instance of this object, which provides configuration information about the available Bluetooth health device, which you must use to perform various operations such as initiate and terminate connections with the BluetoothHealthAPIs.   For more information about using the Bluetooth Health Profile, see the documentation for BluetoothHealth. Accessibility Android 4.0 improves accessibility for sight-impaired users with new explore-by-touch mode and extended APIs that allow you to provide more information about view content or develop advanced accessibility services. Explore-by-touch mode Users with vision loss can now explore the screen by touching and dragging a finger across the screen to hear voice descriptions of the content. Because the explore-by-touch mode works like a virtual cursor, it allows screen readers to identify the descriptive text the same way that screen readers can when the user navigates with a d-pad or trackball—by reading information provided by android:contentDescription and setContentDescription() upon a simulated "hover" event. So, consider this is a reminder that you should provide descriptive text for the views in your application, especially for ImageButton, EditText, ImageView and other widgets that might not naturally contain descriptive text. Accessibility for views To enhance the information available to accessibility services such as screen readers, you can implement new callback methods for accessibility events in your custom View components.  It's important to first note that the behavior of the sendAccessibilityEvent() method has changed in Android 4.0. As with previous version of Android, when the user enables accessibility services on the device and an input event such as a click or hover occurs, the respective view is notified with a call to sendAccessibilityEvent(). Previously, the implementation of sendAccessibilityEvent() would initialize an AccessibilityEvent and send it toAccessibilityManager. The new behavior involves some additional callback methods that allow the view and its parents to add more contextual information to the event:   1. When invoked, the sendAccessibilityEvent() and sendAccessibilityEventUnchecked() methods defer toonInitializeAccessibilityEvent().   Custom implementations of View might want to implement onInitializeAccessibilityEvent() to attach additional accessibility information to the AccessibilityEvent, but should also call the super implementation to provide default information such as the standard content description, item index, and more. However, you should not add additional text content in this callback—that happens next.   1. Once initialized, if the event is one of several types that should be populated with text information, the view then receives a call to dispatchPopulateAccessibilityEvent(), which defers to theonPopulateAccessibilityEvent() callback.   Custom implementations of View should usually implement onPopulateAccessibilityEvent() to add additional text content to the AccessibilityEvent if the android:contentDescription text is missing or insufficient. To add more text description to the AccessibilityEvent, call getText().add().   1. At this point, the View passes the event up the view hierarchy by calling requestSendAccessibilityEvent() on the parent view. Each parent view then has the chance to augment the accessibility information by adding anAccessibilityRecord, until it ultimately reaches the root view, which sends the event to theAccessibilityManager with sendAccessibilityEvent().   In addition to the new methods above, which are useful when extending the View class, you can also intercept these event callbacks on any View by extending AccessibilityDelegate and setting it on the view withsetAccessibilityDelegate(). When you do, each accessibility method in the view defers the call to the corresponding method in the delegate. For example, when the view receives a call to onPopulateAccessibilityEvent(), it passes it to the same method in the View.AccessibilityDelegate. Any methods not handled by the delegate are given right back to the view for default behavior. This allows you to override only the methods necessary for any given view without extending the View class.  If you want to maintain compatibility with Android versions prior to 4.0, while also supporting the new the accessibility APIs, you can do so with the latest version of the v4 support library (in Compatibility Package, r4) using a set of utility classes that provide the new accessibility APIs in a backward-compatible design. Accessibility services If you're developing an accessibility service, the information about various accessibility events has been significantly expanded to enable more advanced accessibility feedback for users. In particular, events are generated based on view composition, providing better context information and allowing accessibility services to traverse view hierarchies to get additional view information and deal with special cases.  If you're developing an accessibility service (such as a screen reader), you can access additional content information and traverse view hierarchies with the following procedure:   1. Upon receiving an AccessibilityEvent from an application, call the AccessibilityEvent.getRecord() to retrieve a specific AccessibilityRecord (there may be several records attached to the event). 2. From either AccessibilityEvent or an individual AccessibilityRecord, you can call getSource() to retrieve aAccessibilityNodeInfo object.   An AccessibilityNodeInfo represents a single node of the window content in a format that allows you to query accessibility information about that node. The AccessibilityNodeInfo object returned from AccessibilityEventdescribes the event source, whereas the source from an AccessibilityRecord describes the predecessor of the event source.   1. With the AccessibilityNodeInfo, you can query information about it, call getParent() or getChild() to traverse the view hierarchy, and even add child views to the node.   In order for your application to publish itself to the system as an accessibility service, it must declare an XML configuration file that corresponds to AccessibilityServiceInfo. For more information about creating an accessibility service, see AccessibilityService and SERVICE\_META\_DATA for information about the XML configuration. Other accessibility APIs If you're interested in the device's accessibility state, the AccessibilityManager has some new APIs such as:   * AccessibilityManager.AccessibilityStateChangeListener is an interface that allows you to receive a callback whenever accessibility is enabled or disabled. * getEnabledAccessibilityServiceList() provides information about which accessibility services are currently enabled. * isTouchExplorationEnabled() tells you whether the explore-by-touch mode is enabled.  Spell Checker Services A new spell checker framework allows apps to create spell checkers in a manner similar to the input method framework (for IMEs). To create a new spell checker, you must implement a service that extends SpellCheckerService and extend the SpellCheckerService.Session class to provide spelling suggestions based on text provided by the interface's callback methods. In the SpellCheckerService.Session callback methods, you must return the spelling suggestions as SuggestionsInfo objects.  Applications with a spell checker service must declare the BIND\_TEXT\_SERVICE permission as required by the service. The service must also declare an intent filter with <action android:name="android.service.textservice.SpellCheckerService" /> as the intent’s action and should include a <meta-data> element that declares configuration information for the spell checker.  See the sample Spell Checker Service app and sample Spell Checker Client app for example code. Text-to-speech Engines Android’s text-to-speech (TTS) APIs have been significantly extended to allow applications to more easily implement custom TTS engines, while applications that want to use a TTS engine have a couple new APIs for selecting an engine. Using text-to-speech engines In previous versions of Android, you could use the TextToSpeech class to perform text-to-speech (TTS) operations using the TTS engine provided by the system or set a custom engine using setEngineByPackageName(). In Android 4.0, thesetEngineByPackageName() method has been deprecated and you can now specify the engine to use with a newTextToSpeech constructor that accepts the package name of a TTS engine.  You can also query the available TTS engines with getEngines(). This method returns a list ofTextToSpeech.EngineInfo objects, which include meta data such as the engine’s icon, label, and package name. Building text-to-speech engines Previously, custom engines required that the engine be built using an undocumented native header file. In Android 4.0, there is a complete set of framework APIs for building TTS engines.  The basic setup requires an implementation of TextToSpeechService that responds to theINTENT\_ACTION\_TTS\_SERVICE intent. The primary work for a TTS engine happens during the onSynthesizeText()callback in a service that extends TextToSpeechService. The system delivers this method two objects:   * SynthesisRequest: This contains various data including the text to synthesize, the locale, the speech rate, and voice pitch. * SynthesisCallback: This is the interface by which your TTS engine delivers the resulting speech data as streaming audio. First the engine must call start() to indicate that the engine is ready to deliver the audio, then callaudioAvailable(), passing it the audio data in a byte buffer. Once your engine has passed all audio through the buffer, call done().   Now that the framework supports a true API for creating TTS engines, support for the native code implementation has been removed. Look for a blog post about a compatibility layer that you can use to convert your old TTS engines to the new framework.  For an example TTS engine using the new APIs, see the Text To Speech Engine sample app. Network Usage Android 4.0 gives users precise visibility of how much network data their applications are using. The Settings app provides controls that allow users to manage set limits for network data usage and even disable the use of background data for individual apps. In order to avoid users disabling your app’s access to data from the background, you should develop strategies to use the data connection efficiently and adjust your usage depending on the type of connection available.  If your application performs a lot of network transactions, you should provide user settings that allow users to control your app’s data habits, such as how often your app syncs data, whether to perform uploads/downloads only when on Wi-Fi, whether to use data while roaming, etc. With these controls available to them, users are much less likely to disable your app’s access to data when they approach their limits, because they can instead precisely control how much data your app uses. If you provide a preference activity with these settings, you should include in its manifest declaration an intent filter for the ACTION\_MANAGE\_NETWORK\_USAGE action. For example:  <activity android:name="DataPreferences" android:label="@string/title\_preferences">      <intent-filter>         <action android:name="android.intent.action.MANAGE\_NETWORK\_USAGE" />         <category android:name="android.intent.category.DEFAULT" />      </intent-filter>  </activity>  This intent filter indicates to the system that this is the activity that controls your application’s data usage. Thus, when the user inspects how much data your app is using from the Settings app, a “View application settings" button is available that launches your preference activity so the user can refine how much data your app uses.  Also beware that getBackgroundDataSetting() is now deprecated and always returns true—usegetActiveNetworkInfo() instead. Before you attempt any network transactions, you should always callgetActiveNetworkInfo() to get the NetworkInfo that represents the current network and query isConnected() to check whether the device has a connection. You can then check other connection properties, such as whether the device is roaming or connected to Wi-Fi. Enterprise Android 4.0 expands the capabilities for enterprise application with the following features. VPN services The new VpnService allows applications to build their own VPN (Virtual Private Network), running as a Service. A VPN service creates an interface for a virtual network with its own address and routing rules and performs all reading and writing with a file descriptor.  To create a VPN service, use VpnService.Builder, which allows you to specify the network address, DNS server, network route, and more. When complete, you can establish the interface by calling establish(), which returns aParcelFileDescriptor.  Because a VPN service can intercept packets, there are security implications. As such, if you implement VpnService, then your service must require the BIND\_VPN\_SERVICE to ensure that only the system can bind to it (only the system is granted this permission—apps cannot request it). To then use your VPN service, users must manually enable it in the system settings. Device policies Applications that manage the device restrictions can now disable the camera using setCameraDisabled() and theUSES\_POLICY\_DISABLE\_CAMERA property (applied with a <disable-camera /> element in the policy configuration file). Certificate management The new KeyChain class provides APIs that allow you to import and access certificates in the system key store. Certificates streamline the installation of both client certificates (to validate the identity of the user) and certificate authority certificates (to verify server identity). Applications such as web browsers or email clients can access the installed certificates to authenticate users to servers. See the KeyChain documentation for more information. Device Sensors Two new sensor types have been added in Android 4.0:   * TYPE\_AMBIENT\_TEMPERATURE: A temperature sensor that provides the ambient (room) temperature in degrees Celsius. * TYPE\_RELATIVE\_HUMIDITY: A humidity sensor that provides the relative ambient (room) humidity as a percentage.   If a device has both TYPE\_AMBIENT\_TEMPERATURE and TYPE\_RELATIVE\_HUMIDITY sensors, you can use them to calculate the dew point and the absolute humidity.  The previous temperature sensor, TYPE\_TEMPERATURE, has been deprecated. You should use theTYPE\_AMBIENT\_TEMPERATURE sensor instead.  Additionally, Android’s three synthetic sensors have been greatly improved so they now have lower latency and smoother output. These sensors include the gravity sensor (TYPE\_GRAVITY), rotation vector sensor (TYPE\_ROTATION\_VECTOR), and linear acceleration sensor (TYPE\_LINEAR\_ACCELERATION). The improved sensors rely on the gyroscope sensor to improve their output, so the sensors appear only on devices that have a gyroscope. Action Bar The ActionBar has been updated to support several new behaviors. Most importantly, the system gracefully manages the action bar’s size and configuration when running on smaller screens in order to provide an optimal user experience on all screen sizes. For example, when the screen is narrow (such as when a handset is in portrait orientation), the action bar’s navigation tabs appear in a “stacked bar," which appears directly below the main action bar. You can also opt-in to a “split action bar," which places all action items in a separate bar at the bottom of the screen when the screen is narrow. Split action bar If your action bar includes several action items, not all of them will fit into the action bar on a narrow screen, so the system will place more of them into the overflow menu. However, Android 4.0 allows you to enable “split action bar" so that more action items can appear on the screen in a separate bar at the bottom of the screen. To enable split action bar, add android:uiOptions with "splitActionBarWhenNarrow" to either your <application> tag or individual<activity> tags in your manifest file. When enabled, the system will add an additional bar at the bottom of the screen for all action items when the screen is narrow (no action items will appear in the primary action bar).  If you want to use the navigation tabs provided by the ActionBar.Tab APIs, but don’t need the main action bar on top (you want only the tabs to appear at the top), then enable the split action bar as described above and also callsetDisplayShowHomeEnabled(false) to disable the application icon in the action bar. With nothing left in the main action bar, it disappears—all that’s left are the navigation tabs at the top and the action items at the bottom of the screen. Action bar styles If you want to apply custom styling to the action bar, you can use new style properties backgroundStacked andbackgroundSplit to apply a background drawable or color to the stacked bar and split bar, respectively. You can also set these styles at runtime with setStackedBackgroundDrawable() and setSplitBackgroundDrawable(). Action provider The new ActionProvider class allows you to create a specialized handler for action items. An action provider can define an action view, a default action behavior, and a submenu for each action item to which it is associated. When you want to create an action item that has dynamic behaviors (such as a variable action view, default action, or submenu), extendingActionProvider is a good solution in order to create a reusable component, rather than handling the various action item transformations in your fragment or activity.  For example, the ShareActionProvider is an extension of ActionProvider that facilitates a “share" action from the action bar. Instead of using traditional action item that invokes the ACTION\_SEND intent, you can use this action provider to present an action view with a drop-down list of applications that handle the ACTION\_SEND intent. When the user selects an application to use for the action, ShareActionProvider remembers that selection and provides it in the action view for faster access to sharing with that app.  To declare an action provider for an action item, include the android:actionProviderClass attribute in the <item>element for your activity’s options menu, with the class name of the action provider as the value. For example:  <item android:id="@+id/menu\_share"        android:title="Share"        android:showAsAction="ifRoom"        android:actionProviderClass="android.widget.ShareActionProvider" />  In your activity’s onCreateOptionsMenu() callback method, retrieve an instance of the action provider from the menu item and set the intent:  public boolean onCreateOptionsMenu(Menu menu) {      getMenuInflater().inflate(R.menu.options, menu);      ShareActionProvider shareActionProvider =            (ShareActionProvider) menu.findItem(R.id.menu\_share).getActionProvider();      // Set the share intent of the share action provider.      shareActionProvider.setShareIntent(createShareIntent());      ...      return super.onCreateOptionsMenu(menu);  }  For an example using the ShareActionProvider, see ActionBarShareActionProviderActivity in ApiDemos. Collapsible action views Action items that provide an action view can now toggle between their action view state and traditional action item state. Previously only the SearchView supported collapsing when used as an action view, but now you can add an action view for any action item and switch between the expanded state (action view is visible) and collapsed state (action item is visible).  To declare that an action item that contains an action view be collapsible, include the “collapseActionView" flag in the android:showAsAction attribute for the <item> element in the menu’s XML file.  To receive callbacks when an action view switches between expanded and collapsed, register an instance ofMenuItem.OnActionExpandListener with the respective MenuItem by calling setOnActionExpandListener(). Typically, you should do so during the onCreateOptionsMenu() callback.  To control a collapsible action view, you can call collapseActionView() and expandActionView() on the respectiveMenuItem.  When creating a custom action view, you can also implement the new CollapsibleActionView interface to receive callbacks when the view is expanded and collapsed. Other APIs for action bar  * setHomeButtonEnabled() allows you to specify whether the icon/logo behaves as a button to navigate home or “up" (pass “true" to make it behave as a button). * setIcon() and setLogo() allow you to define the action bar icon or logo at runtime. * Fragment.setMenuVisibility() allows you to enable or disable the visibility of the options menu items declared by the fragment. This is useful if the fragment has been added to the activity, but is not visible, so the menu items should be hidden. * FragmentManager.invalidateOptionsMenu() allows you to invalidate the activity options menu during various states of the fragment lifecycle in which using the equivalent method from Activity might not be available.  User Interface and Views Android 4.0 introduces a variety of new views and other UI components. GridLayout GridLayout is a new view group that places child views in a rectangular grid. Unlike TableLayout, GridLayout relies on a flat hierarchy and does not make use of intermediate views such as table rows for providing structure. Instead, children specify which row(s) and column(s) they should occupy (cells can span multiple rows and/or columns), and by default are laid out sequentially across the grid’s rows and columns. The GridLayout orientation determines whether sequential children are by default laid out horizontally or vertically. Space between children may be specified either by using instances of the new Space view or by setting the relevant margin parameters on children.  See ApiDemos for samples using GridLayout. TextureView TextureView is a new view that allows you to display a content stream, such as a video or an OpenGL scene. Although similar to SurfaceView, TextureView is unique in that it behaves like a regular view, rather than creating a separate window, so you can treat it like any other View object. For example, you can apply transforms, animate it usingViewPropertyAnimator, or adjust its opacity with setAlpha().  Beware that TextureView works only within a hardware accelerated window.  For more information, see the TextureView documentation. Switch widget The new Switch widget is a two-state toggle that users can drag to one side or the other (or simply tap) to toggle an option between two states.  You can use the android:textOn and android:textOff attributes to specify the text to appear on the switch when in the on and off setting. The android:text attribute also allows you to place a label alongside the switch.  For a sample using switches, see the switches.xml layout file and respective Switches activity. Popup menus Android 3.0 introduced PopupMenu to create short contextual menus that pop up at an anchor point you specify (usually at the point of the item selected). Android 4.0 extends the PopupMenu with a couple useful features:   * You can now easily inflate the contents of a popup menu from an XML menu resource with inflate(), passing it the menu resource ID. * You can also now create a PopupMenu.OnDismissListener that receives a callback when the menu is dismissed.  Preferences A new TwoStatePreference abstract class serves as the basis for preferences that provide a two-state selection option. The new SwitchPreference is an extension of TwoStatePreference that provides a Switch widget in the preference view to allow users to toggle a setting on or off without the need to open an additional preference screen or dialog. For example, the Settings application uses a SwitchPreference for the Wi-Fi and Bluetooth settings. System themes The default theme for all applications that target Android 4.0 (by setting either targetSdkVersion or minSdkVersion to“14" or higher) is now the "device default" theme: Theme.DeviceDefault. This may be the dark Holo theme or a different dark theme defined by the specific device.  The Theme.Holo family of themes are guaranteed to not change from one device to another when running the same version of Android. If you explicitly apply any of the Theme.Holo themes to your activities, you can rest assured that these themes will not change character on different devices within the same platform version.  If you wish for your app to blend in with the overall device theme (such as when different OEMs provide different default themes for the system), you should explicitly apply themes from the Theme.DeviceDefault family. Options menu button Beginning with Android 4.0, you'll notice that handsets no longer require a Menu hardware button. However, there's no need for you to worry about this if your existing application provides an options menu and expects there to be a Menu button. To ensure that existing apps continue to work as they expect, the system provides an on-screen Menu button for apps that were designed for older versions of Android.  For the best user experience, new and updated apps should instead use the ActionBar to provide access to menu items and set targetSdkVersion to "14" to take advantage of the latest framework default behaviors. Controls for system UI visibility Since the early days of Android, the system has managed a UI component known as the status bar, which resides at the top of handset devices to deliver information such as the carrier signal, time, notifications, and so on. Android 3.0 added the system bar for tablet devices, which resides at the bottom of the screen to provide system navigation controls (Home, Back, and so forth) and also an interface for elements traditionally provided by the status bar. In Android 4.0, the system provides a new type of system UI called the navigation bar. You might consider the navigation bar a re-tuned version of the system bar designed for handsets—it provides navigation controls for devices that don’t have hardware counterparts for navigating the system, but it leaves out the system bar's notification UI and setting controls. As such, a device that provides the navigation bar also has the status bar at the top.  To this day, you can hide the status bar on handsets using the FLAG\_FULLSCREEN flag. In Android 4.0, the APIs that control the system bar’s visibility have been updated to better reflect the behavior of both the system bar and navigation bar:   * The SYSTEM\_UI\_FLAG\_LOW\_PROFILE flag replaces the STATUS\_BAR\_HIDDEN flag. When set, this flag enables “low profile" mode for the system bar or navigation bar. Navigation buttons dim and other elements in the system bar also hide. Enabling this is useful for creating more immersive games without distraction for the system navigation buttons. * The SYSTEM\_UI\_FLAG\_VISIBLE flag replaces the STATUS\_BAR\_VISIBLE flag to request the system bar or navigation bar be visible. * The SYSTEM\_UI\_FLAG\_HIDE\_NAVIGATION is a new flag that requests the navigation bar hide completely. Be aware that this works only for the navigation bar used by some handsets (it does **not** hide the system bar on tablets). The navigation bar returns to view as soon as the system receives user input. As such, this mode is useful primarily for video playback or other cases in which the whole screen is needed but user input is not required.   You can set each of these flags for the system bar and navigation bar by calling setSystemUiVisibility() on any view in your activity. The window manager combines (OR-together) all flags from all views in your window and apply them to the system UI as long as your window has input focus. When your window loses input focus (the user navigates away from your app, or a dialog appears), your flags cease to have effect. Similarly, if you remove those views from the view hierarchy their flags no longer apply.  To synchronize other events in your activity with visibility changes to the system UI (for example, hide the action bar or other UI controls when the system UI hides), you should register a View.OnSystemUiVisibilityChangeListener to be notified when the visibility of the system bar or navigation bar changes.  See the OverscanActivity class for a demonstration of different system UI options. Input Framework Android 4.0 adds support for cursor hover events and new stylus and mouse button events. Hover events The View class now supports “hover" events to enable richer interactions through the use of pointer devices (such as a mouse or other devices that drive an on-screen cursor).  To receive hover events on a view, implement the View.OnHoverListener and register it with setOnHoverListener(). When a hover event occurs on the view, your listener receives a call to onHover(), providing the View that received the event and a MotionEvent that describes the type of hover event that occurred. The hover event can be one of the following:   * ACTION\_HOVER\_ENTER * ACTION\_HOVER\_EXIT * ACTION\_HOVER\_MOVE   Your View.OnHoverListener should return true from onHover() if it handles the hover event. If your listener returns false, then the hover event will be dispatched to the parent view as usual.  If your application uses buttons or other widgets that change their appearance based on the current state, you can now use the android:state\_hovered attribute in a state list drawable to provide a different background drawable when a cursor hovers over the view.  For a demonstration of the new hover events, see the Hover class in ApiDemos. Stylus and mouse button events Android now provides APIs for receiving input from a stylus input device such as a digitizer tablet peripheral or a stylus-enabled touch screen.  Stylus input operates in a similar manner to touch or mouse input. When the stylus is in contact with the digitizer, applications receive touch events just like they would when a finger is used to touch the display. When the stylus is hovering above the digitizer, applications receive hover events just like they would when a mouse pointer was being moved across the display when no buttons are pressed.  Your application can distinguish between finger, mouse, stylus and eraser input by querying the “tool type" associated with each pointer in a MotionEvent using getToolType(). The currently defined tool types are: TOOL\_TYPE\_UNKNOWN,TOOL\_TYPE\_FINGER, TOOL\_TYPE\_MOUSE, TOOL\_TYPE\_STYLUS, and TOOL\_TYPE\_ERASER. By querying the tool type, your application can choose to handle stylus input in different ways from finger or mouse input.  Your application can also query which mouse or stylus buttons are pressed by querying the “button state" of aMotionEvent using getButtonState(). The currently defined button states are: BUTTON\_PRIMARY,BUTTON\_SECONDARY, BUTTON\_TERTIARY, BUTTON\_BACK, and BUTTON\_FORWARD. For convenience, the back and forward mouse buttons are automatically mapped to the KEYCODE\_BACK and KEYCODE\_FORWARD keys. Your application can handle these keys to support mouse button based back and forward navigation.  In addition to precisely measuring the position and pressure of a contact, some stylus input devices also report the distance between the stylus tip and the digitizer, the stylus tilt angle, and the stylus orientation angle. Your application can query this information using getAxisValue() with the axis codes AXIS\_DISTANCE, AXIS\_TILT, andAXIS\_ORIENTATION.  For a demonstration of tool types, button states and the new axis codes, see the TouchPaint class in ApiDemos. Properties The new Property class provides a fast, efficient, and easy way to specify a property on any object that allows callers to generically set/get values on target objects. It also allows the functionality of passing around field/method references and allows code to set/get values of the property without knowing the details of what the fields/methods are.  For example, if you want to set the value of field bar on object foo, you would previously do this:  foo.bar = value;  If you want to call the setter for an underlying private field bar, you would previously do this:  foo.setBar(value);  However, if you want to pass around the foo instance and have some other code set the bar value, there is really no way to do it prior to Android 4.0.  Using the Property class, you can declare a Property object BAR on class Foo so that you can set the field on instancefoo of class Foo like this:  BAR.set(foo, value);  The View class now leverages the Property class to allow you to set various fields, such as transform properties that were added in Android 3.0 (ROTATION, ROTATION\_X, TRANSLATION\_X, etc.).  The ObjectAnimator class also uses the Property class, so you can create an ObjectAnimator with a Property, which is faster, more efficient, and more type-safe than the string-based approach. Hardware Acceleration Beginning with Android 4.0, hardware acceleration for all windows is enabled by default if your application has set eithertargetSdkVersion or minSdkVersion to “14" or higher. Hardware acceleration generally results in smoother animations, smoother scrolling, and overall better performance and response to user interaction.  If necessary, you can manually disable hardware acceleration with the hardwareAccelerated attribute for individual<activity> elements or the <application> element. You can alternatively disable hardware acceleration for individual views by calling setLayerType(LAYER\_TYPE\_SOFTWARE).  For more information about hardware acceleration, including a list of unsupported drawing operations, see the Hardware Acceleration document. JNI Changes In previous versions of Android, JNI local references weren’t indirect handles; Android used direct pointers. This wasn't a problem as long as the garbage collector didn't move objects, but it seemed to work because it made it possible to write buggy code. In Android 4.0, the system now uses indirect references in order to detect these bugs.  The ins and outs of JNI local references are described in “Local and Global References" in JNI Tips. In Android 4.0,CheckJNI has been enhanced to detect these errors. Watch the Android Developers Blog for an upcoming post about common errors with JNI references and how you can fix them.  This change in the JNI implementation only affects apps that target Android 4.0 by setting either the targetSdkVersionor minSdkVersion to “14" or higher. If you’ve set these attributes to any lower value, then JNI local references behave the same as in previous versions. WebKit  * WebKit updated to version 534.30 * Support for Indic fonts (Devanagari, Bengali, and Tamil, including the complex character support needed for combining glyphs) in WebView and the built-in Browser * Support for Ethiopic, Georgian, and Armenian fonts in WebView and the built-in Browser * Support for WebDriver makes it easier for you to test apps that use WebView  Android Browser The Browser application adds the following features to support web applications:   * Updated V8 JavaScript compiler for faster performance * Plus other notable enhancements carried over from Android 3.0 are now available for handsets:   + Support for fixed position elements on all pages   + HTML media capture   + Device orientation events   + CSS 3D transformations  Permissions The following are new permissions:   * ADD\_VOICEMAIL: Allows a voicemail service to add voicemail messages to the device. * BIND\_TEXT\_SERVICE: A service that implements SpellCheckerService must require this permission for itself. * BIND\_VPN\_SERVICE: A service that implements VpnService must require this permission for itself. * READ\_PROFILE: Provides read access to the ContactsContract.Profile provider. * WRITE\_PROFILE: Provides write access to the ContactsContract.Profile provider.  Device Features The following are new device features:   * FEATURE\_WIFI\_DIRECT: Declares that the application uses Wi-Fi for peer-to-peer communications.   For a detailed view of all API changes in Android 4.0 (API Level 14), see the API Differences Report. Previous APIs In addition to everything above, Android 4.0 naturally supports all APIs from previous releases. Because the Android 3.x platform is available only for large-screen devices, if you've been developing primarily for handsets, then you might not be aware of all the APIs added to Android in these recent releases.  Here's a look at some of the most notable APIs you might have missed that are now available on handsets as well:  Android 3.0   * Fragment: A framework component that allows you to separate distinct elements of an activity into self-contained modules that define their own UI and lifecycle. See the Fragments developer guide. * ActionBar: A replacement for the traditional title bar at the top of the activity window. It includes the application logo in the left corner and provides a new interface for menu items. See the Action Bar developer guide. * Loader: A framework component that facilitates asynchronous loading of data in combination with UI components to dynamically load data without blocking the main thread. See the Loaders developer guide. * System clipboard: Applications can copy and paste data (beyond mere text) to and from the system-wide clipboard. Clipped data can be plain text, a URI, or an intent. See the Copy and Paste developer guide. * Drag and drop: A set of APIs built into the view framework that facilitates drag and drop operations. See the Drag and Drop developer guide. * An all new flexible animation framework allows you to animate arbitrary properties of any object (View, Drawable, Fragment, Object, or anything else) and define animation aspects such as duration, interpolation, repeat and more. The new framework makes Animations in Android simpler than ever. See the Property Animation developer guide. * RenderScript graphics and compute engine: RenderScript offers a high performance 3D graphics rendering and compute API at the native level, which you write in the C (C99 standard), providing the type of performance you expect from a native environment while remaining portable across various CPUs and GPUs. See the RenderScriptdeveloper guide. * Hardware accelerated 2D graphics: You can now enable the OpenGL renderer for your application by setting {android:hardwareAccelerated="true"} in your manifest element's <application> element or for individual<activity> elements. This results in smoother animations, smoother scrolling, and overall better performance and response to user interaction.   **Note:** If you set your application's minSdkVersion or targetSdkVersion to "14" or higher, hardware acceleration is enabled by default.   * And much, much more. See the Android 3.0 Platform notes for more information.   Android 3.1   * USB APIs: Powerful new APIs for integrating connected peripherals with Android applications. The APIs are based on a USB stack and services that are built into the platform, including support for both USB host and device interactions. See the USB Host and Accessory developer guide. * MTP/PTP APIs: Applications can interact directly with connected cameras and other PTP devices to receive notifications when devices are attached and removed, manage files and storage on those devices, and transfer files and metadata to and from them. The MTP API implements the PTP (Picture Transfer Protocol) subset of the MTP (Media Transfer Protocol) specification. See the android.mtp documentation. * RTP APIs: Android exposes an API to its built-in RTP (Real-time Transport Protocol) stack, which applications can use to manage on-demand or interactive data streaming. In particular, apps that provide VOIP, push-to-talk, conferencing, and audio streaming can use the API to initiate sessions and transmit or receive data streams over any available network. See the android.net.rtp documentation. * Support for joysticks and other generic motion inputs. * See the Android 3.1 Platform notes for many more new APIs.   Android 3.2   * New screens support APIs that give you more control over how your applications are displayed across different screen sizes. The API extends the existing screen support model with the ability to precisely target specific screen size ranges by dimensions, measured in density-independent pixel units (such as 600dp or 720dp wide), rather than by their generalized screen sizes (such as large or xlarge). For example, this is important in order to help you distinguish between a 5" device and a 7" device, which would both traditionally be bucketed as "large" screens. See the blog post, New Tools for Managing Screen Sizes. * New constants for <uses-feature> to declare landscape or portrait screen orientation requirements. * The device "screen size" configuration now changes during a screen orientation change. If your app targets API level 13 or higher, you must handle the "screenSize" configuration change if you also want to handle the"orientation" configuration change. See android:configChanges for more information. * See the Android 3.2 Platform notes for other new APIs.  API Level The Android 4.0 API is assigned an integer identifier—**14**—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 in your application, you need compile the application against an Android platform that supports API level 14 or higher. Depending on your needs, you might also need to add anandroid:minSdkVersion="14" attribute to the <uses-sdk> element. | | |  | | | Honeycomb Welcome to Android 3.0!  The Android 3.0 platform introduces many new and exciting features for users and developers. This document provides a glimpse of some of the new features and technologies, as delivered in Android 3.0. For a more detailed look at new developer APIs, see the Android 3.0 Platform document.   * New User Features * New Developer Features  New User Features http://developer.android.com/sdk/images/3.0/home_hero1.png New UI designed from the ground up for tablets Android 3.0 is a new version of the Android platform that is specifically optimized for devices with larger screen sizes, particularly tablets. It introduces a brand new, truly virtual and “holographic” UI design, as well as an elegant, content-focused interaction model.  Android 3.0 builds on the things people love most about Android — refined multitasking, rich notifications, Home screen customization, widgets, and more — and transforms them with a vibrant, 3D experience and deeper interactivity, making them familiar but even better than before.  The new UI brings fresh paradigms for interaction, navigation, and customization and makes them available to all applications — even those built for earlier versions of the platform. Applications written for Android 3.0 are able to use an extended set of UI objects, powerful graphics, and media capabilities to engage users in new ways.  **System Bar, for global status and notifications**  Across the system and in all applications, users have quick access to notifications, system status, and soft navigation buttons in a System Bar, available at the bottom of the screen. The System Bar is always present and is a key touchpoint for users, but in a new "lights out mode" can also be dimmed for full-screen viewing, such as for videos.  **Action Bar, for application control**  In every application, users have access to contextual options, navigation, widgets, or other types of content in an Action Bar, displayed at the top of the screen. The Action Bar is always present when an application is in use, although its content, theme, and other properties are managed by the application rather than the system. The Action Bar is another key touchpoint for users, especially with action items and an overflow dropdown menu, which users frequently access in a similar manner in most applications.  http://developer.android.com/sdk/images/3.0/homescreen_cust_port.png  **Customizable Home screens**  Five customizable Home screens give users instant access to all parts of the system from any context. Each screen offers a large grid that maintains spatial arrangement in all orientations. Users can select and manipulate Home screen widgets, app shortcuts, and wallpapers using a dedicated visual layout mode. Visual cues and drop shadows improve visibility when adjusting the layout of shortcuts and widgets. Each Home screen also offers a familiar launcher for access to all installed applications, as well as a Search box for universal search of apps, contacts, media files, web content, and more.  http://developer.android.com/sdk/images/3.0/tasks.png  **Recent Apps, for easy visual multitasking**  Multitasking is a key strength of Android and it is central to the Android 3.0 experience. As users launch applications to handle various tasks, they can use the Recent Apps list in the System Bar to see the tasks underway and quickly jump from one application context to another. To help users rapidly identify the task associated with each app, the list shows a snapshot of its actual state when the user last viewed it. Redesigned keyboard The Android soft keyboard is redesigned to make entering text fast and accurate on larger screen sizes. The keys are reshaped and repositioned for improved targeting, and new keys have been added, such as a Tab key, to provide richer and more efficient text input. Users can touch-hold keys to access menus of special characters and switch text/voice input modes from a button in the System Bar.  http://developer.android.com/sdk/images/3.0/copy.png Improved text selection, copy and paste When entering or viewing text, a new UI lets users quickly select a word by press-hold and then adjust the selection area as needed by dragging a set of bounding arrows to new positions. Users can then select an action from the Action Bar, such as copy to the clipboard, share, paste, web search, or find. New connectivity options Android 3.0 includes new connectivity features that add versatility and convenience for users. Built-in support for Media/Picture Transfer Protocol lets users instantly sync media files with a USB-connected camera or desktop computer, without needing to mount a USB mass-storage device. Users can also connect full keyboards over either USB or Bluetooth, for a familiar text-input environment. For improved wi-fi connectivity, a new combo scan reduces scan times across bands and filters. New support for Bluetooth tethering means that more types of devices can share the network connection of an Android-powered device. Updated set of standard apps http://developer.android.com/sdk/images/3.0/browser.png http://developer.android.com/sdk/images/3.0/camera.png  The Android 3.0 platform includes an updated set of standard applications that are designed for use on larger screen devices. The sections below highlight some of the new features.  **Browser**    The browser includes new features that let users navigate and organize more efficiently. Multiple tabs replace browser windows and a new “incognito” mode allows anonymous browsing. Bookmarks and history are presented and managed in a single unified view. Users can now choose to automatically sign into Google sites on the browser with a supplied account and sync bookmarks with Google Chrome. New multitouch support is now available to JavaScript and plugins. Users can enjoy a better browsing experience at non-mobile sites through an improved zoom and viewport model, overflow scrolling, support for fixed positioning, and more.  **Camera and Gallery**  The Camera application has been redesigned to take advantage of a larger screen for quick access to exposure, focus, flash, zoom, front-facing camera, and more. To let users capture scenes in new ways, it adds built-in support for time-lapse video recording. The Gallery application lets users view albums and other collections in full-screen mode, with easy access to thumbnails for other photos in the collection.  **Contacts**  The Contacts app uses a new two-pane UI and Fast Scroll to let users easily organize and locate contacts. The application offers improved formatting of international phone numbers as user types, based on home country and an international number parsing library. Contact information is presented in a card-like UI, making it easier for users to read and edit contacts.  **Email**  The Email application uses a new two-pane UI to make viewing and organizing messages more efficient. The app lets users select one or more messages, then select an action from the Action Bar, such as moving them to a folder. Users can sync attachments for later viewing and keep track of email using a home screen Widget. New Developer Features The Android 3.0 platform is designed specially to meet the unique needs of applications on devices with larger screen sizes. It offers all of the tools developers need to create incredible visual and interaction experiences on these devices.   * New UI framework for creating great tablet apps * High-performance 2D and 3D graphics * Support for multicore processor architectures * Rich multimedia and connectivity * Enhancements for enterprise * Compatibility with existing apps  New UI Framework for creating great tablet apps http://developer.android.com/sdk/images/3.0/contacts.png  **Activity fragments, for greater control of content and design flexibility**  Starting with Android 3.0, developers can break the Activities of their applications into subcomponents called Fragments, then combine them in a variety of ways to create a richer, more interactive experience. For example, an application can use a set of Fragments to create a true multipane UI, with the user being able to interact with each pane independently. Fragments can be added, removed, replaced, and animated inside an Activity dynamically, and they are modular and reusable across multiple Activities. Because they are modular, Fragments also offer an efficient way for developers to write applications that can run properly on both larger screen as well as smaller screen devices.  **Redesigned UI widgets**  Android 3.0 offers an updated set of UI widgets that developers can use to quickly add new types of content to their applications. The new UI widgets are redesigned for use on larger screens such as tablets and incorporate the new holographic UI theme. Several new widget types are available, including a 3D stack, search box, a date/time picker, number picker, calendar, popup menu, and others. Most of the redesigned UI widgets can now be used as remote views in application widgets displayed on the home screen. Applications written for earlier versions can inherit the new Widget designs and themes.  http://developer.android.com/sdk/images/3.0/widgets.png  **Expanded Home screen widgets**  Home screen widgets are popular with users because they offer fast access to application-specific data directly from the home screen. Android 3.0 lets developers take home screen widgets to the next level, offering more types of content and new modes of interaction with users. Developers can now use more standard UI widget types home screen widgets, including widgets that let users flip through collections of content as 3D stacks, grids, or lists. Users can interact with the home screen widgets in new ways, such as by using touch gestures to scroll and flip the content displayed in a widget.  **Persistent Action Bar**  The platform provides each application with its own instance of the Action Bar at the top of the screen, which the application can use to give the user quick access to contextual options, widgets, status, navigation, and more. The application can also customize the display theme of its Action Bar instance. The Action Bar lets developers expose more features of their applications to users in a familiar location, while also unifying the experience of using an application that spans multiple Activities or states.  **Richer notifications**  Notifications are a key part of the Android user experience because they let applications show key updates and status information to users in real time. Android 3.0 extends this capability, letting developers include richer content and control more properties. A new builder class lets developers quickly create notifications that include large and small icons, a title, a priority flag, and any properties already available in previous versions. Notifications can offer more types of content by building on the expanded set of UI Widgets that are now available as remote Views.  http://developer.android.com/sdk/images/3.0/mail_drag.png  **Multiselect, clipboard, and drag-and-drop**  The platform offers convenient new interaction modes that developers can use. For managing collections of items in lists or grids, developers can offer a new multiselect mode that lets users choose multiple items for an action. Developers can also use a new system-wide Clipboard to let users easily copy any type of data into and out of their applications. To make it easier for users to manage and organize files, developers can now add drag-and-drop interaction through a DragEvent framework. High-performance 2D and 3D graphics **New animation framework**  The platform includes a flexible new animation framework that lets developers easily animate the properties of UI elements such as Views, Widgets, Fragments, Drawables, or any arbitrary object. Animations can create fades or movement between states, loop an animated image or an existing animation, change colors, and much more. Adding animation to UI elements can add visual interest to an application and refine the user experience, to keep users engaged.  **Hardware-accelerated 2D graphics**  Android 3.0 offers a new hardware-accelerated OpenGL renderer that gives a performance boost to many common graphics operations for applications running in the Android framework. When the renderer is enabled, most operations in Canvas, Paint, Xfermode, ColorFilter, Shader, and Camera are accelerated. Developers can control how hardware-acceleration is applied at every level, from enabling it globally in an application to enabling it in specific Activities and Views inside the application.  **Renderscript 3D graphics engine**  Renderscript is a runtime 3D framework that provides both an API for building 3D scenes as well as a special, platform-independent shader language for maximum performance. Using Renderscript, you can accelerate graphics operations and data processing. Renderscript is an ideal way to create high-performance 3D effects for applications, wallpapers, carousels, and more. Support for multicore processor architectures Android 3.0 is the first version of the platform designed to run on either single or multicore processor architectures. A variety of changes in the Dalvik VM, Bionic library, and elsewhere add support for symmetric multiprocessing in multicore environments. These optimizations can benefit all applications, even those that are single-threaded. For example, with two active cores, a single-threaded application might still see a performance boost if the Dalvik garbage collector runs on the second core. The system will arrange for this automatically. Rich multimedia and connectivity **HTTP Live streaming**  Applications can now pass an M3U playlist URL to the media framework to begin an HTTP Live streaming session. The media framework supports most of the HTTP Live streaming specification, including adaptive bit rate.  **Pluggable DRM framework**  Android 3.0 includes an extensible DRM framework that lets applications manage protected content according to a variety of DRM mechanisms that may be available on the device. For application developers, the framework API offers an consistent, unified API that simplifies the management of protected content, regardless of the underlying DRM engines.  **Digital media file transfer**  The platform includes built-in support for Media/Picture Transfer Protocol (MTP/PTP) over USB, which lets users easily transfer any type of media files between devices and to a host computer. Developers can build on this support, creating applications that let users create or manage media files that they may want to transfer or share across devices.  **More types of connectivity**  The platform offers new connectivity that developers can build on. API support for Bluetooth A2DP and HSP profiles lets applications query Bluetooth profiles for connected devices, audio state, and more, then notify the user. For example, a music application can check connectivity and status and let the user know that music is playing through a stereo headset. Applications can also register to receive system broadcasts of pre-defined vendor-specific AT commands, such as Platronics Xevent. For example, an application could receive broadcasts that indicate a connected device's battery level and could notify the user or take other action as needed. Applications can also take advantage of the platform's new support for full keyboards connected by USB or Bluetooth. Enhancements for enterprise In Android 3.0, developers of device administration applications can support new types of policies, including policies for encrypted storage, password expiration, password history, and password complex characters required. Compatibility with existing apps Android 3.0 brings a new UI designed for tablets and other larger screen devices, but it also is fully compatible with applications developed for earlier versions of the platform, or for smaller screen sizes. Existing applications can seamlessly participate in the new holographic UI theme without code changes, by adding a single attribute in their manifest files. The platform emulates the Menu key, which is replaced by the overflow menu in the Action Bar in the new UI. Developers wanting to take fuller advantage of larger screen sizes can also create dedicated layouts and assets for larger screens and add them to their existing applications. Android 3.2 APIs API Level: **13**  Android 3.2 (HONEYCOMB\_MR2) is an incremental platform release that adds new capabilities for users and developers. The sections below provide an overview of the new features and developer APIs.  For developers, the Android 3.2 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 3.2, use the Android SDK Manager to download the platform into your SDK. Platform HighlightsNew user features  * **Optimizations for a wider range of tablets**   Android 3.2 includes a variety of optimizations across the system to ensure a great user experience on a wider range of tablet devices.   * **Compatibility zoom for fixed-sized apps**   Android 3.2 introduces a new compatibility zoom mode that gives users a new way to view fixed-sized apps on larger devices. The new mode provides a pixel-scaled alternative to the standard UI stretching for apps that are not designed to run on larger screen sizes, such as on tablets. The new mode is accessible to users from a menu icon in the system bar, for apps that need compatibility support.   * **Media sync from SD card**   On devices that support an SD card, users can now load media files directly from the SD card to apps that use them. A system facility makes the files accessible to apps from the system media store. New developer features  * **Extended API for managing screens support**   Android 3.2 introduces extensions to the platform's screen support API to give developers additional ways to manage application UI across the range of Android-powered devices. The API includes new resource qualifiers and new manifest attributes that give you more precise control over how your apps are displayed on different sizes, rather than relying on generalized size categories.  To ensure the best possible display for fixed-sized apps and apps with limited support for various screen sizes, the platform also provides a new zoom compatibility mode that renders the UI on a smaller screen area, then scales it up to fill the space available on the display. For more information about the screen support API and the controls it provides, see the sections below. API OverviewScreens Support APIs Android 3.2 introduces new screens support APIs that give you more control over how their applications are displayed across different screen sizes. The API builds on the existing screens-support API, including the platform's generalized screen density model, but extends it with the ability to precisely target specific screen ranges by their dimensions, measured in density-independent pixel units (such as 600dp or 720dp wide), rather than by their generalized screen sizes (such as large or xlarge)  When designing an application's UI, you can still rely on the platform to provide density abstraction, which means that applications do not need to compensate for the differences in actual pixel density across devices. You can design the application UI according to the amount of horizontal or vertical space available. The platform expresses the amount of space available using three new characteristics: smallestWidth, width, and height.   * A screen's smallestWidth is its fundamental minimum size, measured in density-independent pixel ("dp") units. Of the screen's height or width, it is the shorter of the two. For a screen in portrait orientation, the smallestWidth is normally based on its width, while in landscape orientation it is based on its height. In all cases, the smallestWidth is derived from a fixed characteristic of the screen and the value does not change, regardless of orientation. The smallestWidth is important for applications because it represents the shortest possible width in which the application UI will need to be drawn, not including screen areas reserved by the system. * In contrast, a screen's width and height represent the current horizontal or vertical space available for application layout, measured in "dp" units, not including screen areas reserved by the system. The width and height of a screen change when the user switches orientation between landscape and portrait.   The new screens support API is designed to let you manage application UI according to the smallestWidth of the current screen. You can also manage the UI according to current width or height, as needed. For those purposes, the API provides these tools:   * New resource qualifiers for targeting layouts and other resources to a minimum smallestWidth, width, or height, and * New manifest attributes, for specifying the app's maximum screen compatibility range   Additionally, applications can still query the system and manage UI and resource loading at runtime, as in the previous versions of the platform.  Since the new API lets you target screens more directly through smallestWidth, width, and height, it's helpful to understand the typical characteristics of the different screen types. The table below provides some examples, measured in "dp" units.  **Table 1.** Typical devices, with density and size in dp.   |  |  |  |  | | --- | --- | --- | --- | | Type | Density (generalized) | Dimensions (dp) | smallestWidth (dp) | | Baseline phone | mdpi | 320x480 | 320 | | Small tablet/large phone | mdpi | 480x800 | 480 | | 7-inch tablet | mdpi | 600x1024 | 600 | | 10-inch tablet | mdpi | 800x1280 | 800 |   The sections below provide more information about the new screen qualifiers and manifest attributes. For complete information about how to use the screen support API, see Supporting Multiple Screens. New resource qualifiers for screens support The new resource qualifiers in Android 3.2 let you better target your layouts for ranges of screen sizes. Using the qualifiers, you can create resource configurations designed for a specific minimum smallestWidth, current width, or current height, measured in density-independent pixels.  The new qualifiers are:   * swNNNdp — Specifies the minimum smallestWidth on which the resource should be used, measured in "dp" units. As mentioned above, a screen's smallestWidth is constant, regardless of orientation. Examples: sw320dp, sw720dp,sw720dp. * wNNNdp and hNNNdp — Specifies the minimum width or height on which the resource should be used, measured in "dp" units. As mentioned above, a screen's width and height are relative to the orientation of the screen and change whenever the orientation changes. Examples: w320dp, w720dp, h1024dp.     You can also create multiple overlapping resource configurations if needed. For example, you could tag some resources for use on any screen wider than 480 dp, others for wider than 600 dp, and others for wider than 720 dp. When multiple resource configurations are qualified for a given screen, the system selects the configuration that is the closest match. For precise control over which resources are loaded on a given screen, you can tag resources with one qualifier or combine several new or existing qualifiers.  Based on the typical dimensions listed earlier, here are some examples of how you could use the new qualifiers:  res/layout/main\_activity.xml   # For phones  res/layout-sw600dp/main\_activity.xml   # For 7” tablets  res/layout-sw720dp/main\_activity.xml   # For 10” tablets  res/layout-w600dp/main\_activity.xml   # Multi-pane when enough width  res/layout-sw600dp-w720dp/main\_activity.xml   # For large width  Older versions of the platform will ignore the new qualifiers, so you can mix them as needed to ensure that your app looks great on any device. Here are some examples:  res/layout/main\_activity.xml   # For phones  res/layout-xlarge/main\_activity.xml   # For pre-3.2 tablets  res/layout-sw600dp/main\_activity.xml   # For 3.2 and up tablets  For complete information about how to use the new qualifiers, see Using new size qualifiers. New manifest attributes for screen-size compatibility The framework offers a new set of <supports-screens> manifest attributes that let you manage your app's support for different screen sizess. Specifically, you can specify the largest and smallest screens on which your app is designed to run, as well as the largest screen on which it is designed run without needing the system's new screen compatibility mode. Like the resource qualifiers described above, the new manifest attributes specify the range of screens that the application supports, as specified by the smallestWidth.  The new manifest attributes for screen support are:   * android:compatibleWidthLimitDp="numDp" — This attribute lets you specify the maximum smallestWidth on which the application can run without needing compatibility mode. If the current screen is larger than the value specified, the system displays the application in normal mode but allows the user to optionally switch to compatibility mode through a setting in the system bar. * android:largestWidthLimitDp="numDp" — This attribute lets you specify the maximum smallestWidth on which the application is designed to run. If the current screen is larger than the value specified, the system forces the application into screen compatibility mode, to ensure best display on the current screen. * android:requiresSmallestWidthDp="numDp" — This attribute lets you specify the minimum smallestWidth on which the application can run. If the current screen is smaller than the value specified, the system considers the application incompatible with the device, but does not prevent it from being installed and run.   **Note:** Google Play does not currently filter apps based on any of the attributes above. Support for filtering will be added in a later platform release. Applications that require filtering based on screen size can use the existing <supports-screens> attributes.  For complete information about how to use the new attributes, see Declaring screen size support. Screen compatibility mode Android 3.2 provides a new screen compatibility mode for applications explicitly declaring that they do not support screens as large as the one on which they are running. This new "zoom" mode is a pixel-scaled — it renders the application in a smaller screen area and then scales the pixels to fill the current screen.  By default, the system offers screen compatibility mode as an user option, for apps that require it. Users can turn the zoom mode on and off using a control available in the system bar.  Because the new screen compatibility mode may not be appropriate for all applications, the platform allows the application to disable it using manifest attributes. When disabled by the app, the system does not offer "zoom" compatibility mode as an option for users when the app is running.  **Note:** For important information about how to control compatibility mode in your applications, please review the New Mode for Apps on Large Screens article on the Android Developers Blog. New screen density for 720p televisions and similar devices To meet the needs of applications running on 720p televisions or similar with moderate density screens, Android 3.2 introduces a new generalized density, tvdpi, with an approximate dpi of 213. Applications can query for the new density in densityDpi and can use the new tvdpi qualifier to tag resources for televisions and similar devices. For example:  res/drawable-tvdpi/my\_icon.png   # Bitmap for tv density  In general, applications should not need to work with this density. For situations where output is needed for a 720p screen, the UI elements can be scaled automatically by the platform. UI framework  * Fragments   + New Fragment.SavedState class holds the state information retrieved from a fragment instance throughsaveFragmentInstanceState().   + New method saveFragmentInstanceState() saves the current instance state of the given Fragment. The state can be used later when creating a new instance of the Fragment that matches the current state.   + New method setInitialSavedState() sets the initial saved state for a Fragment when first constructed.   + New onViewCreated() callback method notifies the Fragment that onCreateView() has returned, but before any saved state has been restored in to the View.   + isDetached() method determines whether the Fragment has been explicitly detached from the UI.   + New attach() and detach() methods let an application re-attach or detach fragments in the UI.   + A new setCustomAnimations() overload method lets you set specific animation resources to run for enter/exit operations and specifically when popping the back stack. The existing implementation does not account for the different behavior of fragments when popping the back stack. * Screen size information in ActivityInfo and ApplicationInfo   + ActivityInfo adds CONFIG\_SCREEN\_SIZE and CONFIG\_SMALLEST\_SCREEN\_SIZE as bit masks inconfigChanges. The bits indicate whether an Activity can itself handle the screen size and smallest screen size.   + ApplicationInfo adds largestWidthLimitDp, compatibleWidthLimitDp, and requiresSmallestWidthDpfields, derived from the corresponding <supports-screens> attributes in the application manifest file. * Helpers for getting display size from WindowManager   + New methods getSize() and getRectSize() let applications get the raw size of the display. * New public "holographic" styles   + The platform now exposes a variety of public "holographic" styles for text, actionbar widgets and tabs, and more. See R.style for a full list. * LocalActivityManager, ActivityGroup, and LocalActivityManager are now deprecated   + New applications should use Fragments instead of these classes. To continue to run on older versions of the platform, you can use the v4 Support Library (compatibility library), available in the Android SDK. The v4 Support Library provides a version of the Fragment API that is compatible down to Android 1.6 (API level 4).   + For apps developing against Android 3.0 (API level 11) or higher, tabs are typically presented in the UI using the newActionBar.newTab() and related APIs for placing tabs within their action bar area.    Media framework  * Applications that use the platform's media provider (MediaStore) can now read media data directly from the removeable SD card, where supported by the device. Applications can also interact with the SD card files directly, using the MTP API.  Graphics  * Parcelable utilities in Point and PointF   + Point and PointF classes now include the Parcelable interface and utility methods describeContents(),readFromParcel(), and writeToParcel().  IME framework  * New getModifiers() method for retrieving the current state of the modifier keys.  USB framework  * New getRawDescriptors() method for retrieving the raw USB descriptors for the device. You can use the method to access descriptors not supported directly via the higher level APIs.  Network  * Network type constants   + ConnectivityManager adds the constants TYPE\_ETHERNET and TYPE\_BLUETOOTH.  Telephony  * New NETWORK\_TYPE\_HSPAP network type constant.  Core utilities  * Parcelable utilities   + New interface Parcelable.ClassLoaderCreator allows the application to receive the ClassLoader in which the object is being created.   + New adoptFd, dup(), and fromFd() for managing ParcelFileDescriptor objects. * Binder and IBinder   + New method dumpAsync() in Binder and IBinder let applications dump to a specified file, ensuring that the target executes asynchronously.   + New IBinder protocol transaction code TWEET\_TRANSACTION lets applications send a tweet to the target object.  New feature constants The platform adds new hardware feature constants that you can declare in their application manifests, to inform external entities such as Google Play of required hardware and software capabilities. You declare these and other feature constants in <uses-feature> manifest elements.  Google Play filters applications based on their <uses-feature> attributes, to ensure that they are available only to devices on which their requirements are met.   * Feature constants for landscape or portrait requirements   Android 3.2 introduces new feature constants that let applications specify whether they require display in landscape orientation, portrait orientation, or both. Declaring these constants indicates that the application must not be installed on a device that doesn't offer the associated orientation. Conversely, if one or both of the constants are not declared, it indicates that the application does not have a preference for the undeclared orientations and may be installed on a device that doesn't offer them.   * + android.hardware.screen.landscape — The application requires display in landscape orientation.   + android.hardware.screen.portrait — The application requires display in portrait orientation.   A typical application that functions properly in both landscape and portrait orientations would not normally need to declare an orientation requirement. Rather, an application designed primarily for one orientation, such as an app designed for a television, could declare one of the constants to ensure that it isn't available to devices that don't provide that orientation.  If any of activities declared in the manifest request that they run in a specific orientation, using theandroid:screenOrientation attribute, then this also declares that the application requires that orientation.   * Other feature constants   + android.hardware.faketouch.multitouch.distinct — The application requires support for emulated mulitouch input with distinct tracking of two or more points.   + android.hardware.faketouch.multitouch.jazzhand — The application requires support for emulated mulitouch input with distinct tracking of five or more points.  API Differences Report For a detailed view of all API changes in Android 3.2 (API Level 13), see the API Differences Report. API Level The Android 3.2 platform delivers an updated version of the framework API. The Android 3.2 API is assigned an integer identifier — **13** — 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 3.2 in your application, you need compile the application against the Android library that is provided in the Android 3.2 SDK platform. Depending on your needs, you might also need to add anandroid:minSdkVersion="13" attribute to the <uses-sdk> element in the application's manifest.   |  | | --- | | Android 3.1 APIs API Level: **12**  For developers, the Android 3.1 platform (HONEYCOMB\_MR1) 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. The downloadable platform includes no external libraries.  For developers, the Android 3.1 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 3.1, use the Android SDK Manager to download the platform into your SDK. API Overview The sections below provide a technical overview of what's new for developers in Android 3.1, including new features and changes in the framework API since the previous version. USB APIs Android 3.1 introduces powerful new APIs for integrating connected peripherals with applications running on the platform. The APIs are based on a USB (Universal Serial Bus) stack and services that are built into the platform, including support for both USB host and device interactions. Using the APIs, developers can create applications that are able to discover, communicate with, and manage a variety of device types connected over USB.  The stack and APIs distinguish two basic types of USB hardware, based on whether the Android-powered device is acting as host or the external hardware is acting as host:   * A USB device is a piece of connected hardware that depends on the Android-powered device to serve as host. For example, most input devices, mice, and joysticks are USB devices, as are many cameras, hubs, and so on. * A USB accessory is a piece of connected hardware that has a USB host controller, provides power, and is designed to communicate with Android-powered devices over USB, A variety of peripherals can connect as accessories, from robotics controllers to musical equipment, exercise bicycles, and more.   For both types — USB devices and USB accessories — the platform's USB APIs support discovery by intent broadcast when attached or detached, as well as standard interfaces, endpoints, and transfer modes (control, bulk, and interrupt).  The USB APIs are available in the package android.hardware.usb. The central class is UsbManager, which provides helper methods for identifying and communicating with both USB devices and USB accessories. Applications can acquire an instance of UsbManager and then query for the list of attached devices or accessories and then communicate with or manage them. UsbManager also declares intent actions that the system broadcasts, to announce when a USB device or accessory is attached or detached.  Other classes include:   * UsbDevice, a class representing external hardware connected as a USB device (with the Android-powered device acting as host). * UsbAccessory, representing external hardware connected as the USB host (with the Android-powered device acting as a USB device). * UsbInterface and UsbEndpoint, which provide access to standard USB interfaces and endpoints for a device. * UsbDeviceConnection and UsbRequest, for sending and receiving data and control messages to or from a USB device, sychronously and asynchronously. * UsbConstants, which provides constants for declaring endpoint types, device classes, and so on.   Note that although the USB stack is built into the platform, actual support for USB host and open accessory modes on specific devices is determined by their manufacturers. In particular, host mode relies on appropriate USB controller hardware in the Android-powered device.  Additionally, developers can request filtering on Google Play, such that their applications are not availabe to users whose devices do not provide the appropriate USB support. To request filtering, add one or both of the elements below to the application manifest, as appropriate:   * If the application should only be visible to devices that support USB host mode (connection of USB devices), declare this element:   <uses-feature android:name="android.hardware.usb.host" android:required="true">   * If the application should only be visible to devices that support USB accessories (connection of USB hosts), declare this element:   <uses-feature android:name="android.hardware.usb.accessory" android:required="true">  For complete information about how to develop applications that interact with USB accessories, please see the developer documentation.  To look at sample applications that use the USB host API, see ADB Test and Missile Launcher MTP/PTP API Android 3.1 exposes a new MTP API that lets applications interact directly with connected cameras and other PTP devices. The new API makes it easy for an application to receive notifications when devices are attached and removed, manage files and storage on those devices, and transfer files and metadata to and from them. The MTP API implements the PTP (Picture Transfer Protocol) subset of the MTP (Media Transfer Protocol) specification.  The MTP API is available in the android.mtp package and provides these classes:   * The MtpDevice encapsulates an MTP device that is connected over the USB host bus. An application can instantiate an object of this type and then use its methods to get information about the device and objects stored on it, as well as opening the connection and transferring data. Some of the methods include:   + getObjectHandles() returns a list of handles for all objects on the device that match a specified format and parent. To get information about an object, an application can pass a handle to getObjectInfo().   + importFile() lets an application copy data for an object to a file in external storage. This call may block for an arbitrary amount of time depending on the size of the data and speed of the devices, so should be made from a spearate thread.   + open() lets an application open a connected MTP/PTP device.   + getThumbnail() returns the thumbnail of the object as a byte array. * MtpStorageInfo holds information about about a storage unit on an MTP device, corresponding to the StorageInfo Dataset described in section 5.2.2 of the MTP specification. Methods in the class let an application get a storage unit’s description string, free space, maximum storage capacity, storage ID, and volume identifier. * MtpDeviceInfo holds information about an MTP device corresponding to the DeviceInfo Dataset described in section 5.1.1 of the MTP specification. Methods in the class let applications get a device’s manufacturer, model, serial number, and version. * MtpObjectInfo holds information about an object stored on an MTP device, corresponding to the ObjectInfo Dataset described in section 5.3.1 of the MTP specification. Methods in the class let applications get an object’s size, data format, association type, creation date, and thumbnail information. * MtpConstants provides constants for declaring MTP file format codes, association type, and protection status.  Support for new input devices and motion events Android 3.1 extends the input subsystem to support new input devices and new types of motion events, across all views and windows. Developers can build on these capabilities to let users interact with their applications using mice, trackballs, joysticks, gamepads, and other devices, in addition to keyboards and touchscreens.  For handling mouse, scrollwheel, and trackball input, the platform supports two new motion event actions:   * ACTION\_SCROLL, which describes the pointer location at which a non-touch scroll motion, such as from a mouse scroll wheel, took place. In the MotionEvent, the value of the AXIS\_HSCROLL and AXIS\_VSCROLL axes specify the relative scroll movement. * ACTION\_HOVER\_MOVE, reports the current position of the mouse when no buttons are pressed, as well as any intermediate points since the last HOVER\_MOVE event. Hover enter and exit notifications are not yet supported.   To support joysticks and gamepads, the InputDevice class includes these new input device sources:   * SOURCE\_CLASS\_JOYSTICK — the source device has joystick axes. * SOURCE\_CLASS\_BUTTON — the source device has buttons or keys. * SOURCE\_GAMEPAD — the source device has gamepad buttons such as KEYCODE\_BUTTON\_A or KEYCODE\_BUTTON\_B. Implies SOURCE\_CLASS\_BUTTON * SOURCE\_JOYSTICK — the source device has joystick axes. Implies SOURCE\_CLASS\_JOYSTICK.   To describe motion events from these new sources, as well as those from mice and trackballs, the platform now defines axis codes on MotionEvent, similar to how it defines key codes on KeyEvent. New axis codes for joysticks and game controllers include AXIS\_HAT\_X, AXIS\_HAT\_Y, AXIS\_RTRIGGER, AXIS\_ORIENTATION, AXIS\_THROTTLE, and many others. Existing MotionEvent axes are represented by AXIS\_X, AXIS\_Y, AXIS\_PRESSURE, AXIS\_SIZE, AXIS\_TOUCH\_MAJOR,AXIS\_TOUCH\_MINOR, AXIS\_TOOL\_MAJOR, AXIS\_TOOL\_MINOR, and AXIS\_ORIENTATION.  Additionally, MotionEvent defines a number of generic axis codes that are used when the framework does not know how to map a particular axis. Specific devices can use the generic axis codes to pass custom motion data to applications. For a full list of axes and their intended interpretations, see the MotionEvent class documentation.  The platform provides motion events to applications in batches, so a single event may contain a current position and multiple so-called historical movements. Applications should use getHistorySize() to get the number of historical samples, then retrieve and process all historical samples in order using getHistoricalAxisValue(). After that, applications should process the current sample using getAxisValue().  Some axes can be retrieved using special accessor methods. For example, instead of calling getAxisValue(), applications can call getX(). Axes that have built-in accessors include AXIS\_X, AXIS\_Y, AXIS\_PRESSURE, AXIS\_SIZE,AXIS\_TOUCH\_MAJOR, AXIS\_TOUCH\_MINOR, AXIS\_TOOL\_MAJOR, AXIS\_TOOL\_MINOR, and AXIS\_ORIENTATION.  Each input device has a unique, system-assigned ID and may also provide multiple sources. When a device provides multiple sources, more than one source can provide axis data using the same axis. For example, a touch event coming from the touch source uses the X axis for screen position data, while a joystick event coming from the joystick source will use the X axis for the stick position instead. For this reason, it's important for applications to interpret axis values according to the source from which they originate. When handling a motion event, applications should use methods on the InputDevice class to determine the axes supported by a device or source. Specifically, applications can usegetMotionRanges() to query for all axes of a device or all axes of a given source of the device. In both cases, the range information for axes returned in the InputDevice.MotionRange object specifies the source for each axis value.  Finally, since the motion events from joysticks, gamepads, mice, and trackballs are not touch events, the platform adds a new callback method for passing them to a View as "generic" motion events. Specifically, it reports the non-touch motion events to Views through a call to onGenericMotionEvent(), rather than to onTouchEvent().  The platform dispatches generic motion events differently, depending on the event source class.SOURCE\_CLASS\_POINTER events go to the View under the pointer, similar to how touch events work. All others go to the currently focused View. For example, this means a View must take focus in order to receive joystick events. If needed, applications can handle these events at the level of Activity or Dialog by implementing onGenericMotionEvent() there instead.  To look at a sample application that uses joystick motion events, see GameControllerInput and GameView. RTP API Android 3.1 exposes an API to its built-in RTP (Real-time Transport Protocol) stack, which applications can use to manage on-demand or interactive data streaming. In particular, apps that provide VOIP, push-to-talk, conferencing, and audio streaming can use the API to initiate sessions and transmit or receive data streams over any available network.  The RTP API is available in the android.net.rtp package. Classes include:   * RtpStream, the base class of streams that send and receive network packets with media payloads over RTP. * AudioStream, a subclass of RtpStream that carries audio payloads over RTP. * AudioGroup, a local audio hub for managing and mixing the device speaker, microphone, and AudioStream. * AudioCodec, which holds a collection of codecs that you define for an AudioStream.   To support audio conferencing and similar usages, an application instantiates two classes as endpoints for the stream:   * AudioStream specifies a remote endpoint and consists of network mapping and a configured AudioCodec. * AudioGroup represents the local endpoint for one or more AudioStreams. The AudioGroup mixes all theAudioStreams and optionally interacts with the device speaker and the microphone at the same time.   The simplest usage involves a single remote endpoint and local endpoint. For more complex usages, please refer to the limitations described for AudioGroup.  To use the RTP API, applications must request permission from the user by declaring <uses-permission android:name="android.permission.INTERNET"> in their manifest files. To acquire the device microphone, the<uses-permission android:name="android.permission.RECORD\_AUDIO"> permission is also required. Resizable app widgets Starting in Android 3.1, developers can make their homescreen widgets resizeable — horizontally, vertically, or on both axes. Users touch-hold a widget to show its resize handles, then drag the horizontal and/or vertical handles to change the size on the layout grid.  Developers can make any Home screen widget resizeable by defining a resizeMode attribute in the widget'sAppWidgetProviderInfo metadata. Values for the resizeMode attribute include "horizontal", "vertical", and "none". To declare a widget as resizeable horizontally and vertically, supply the value "horizontal|vertical".  Here's an example:  <appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"      android:minWidth="294dp"      android:minHeight="72dp"      android:updatePeriodMillis="86400000"      android:previewImage="@drawable/preview"      android:initialLayout="@layout/example\_appwidget"      android:configure="com.example.android.ExampleAppWidgetConfigure"      android:resizeMode="horizontal|vertical" >  </appwidget-provider>  For more information about Home screen widgets, see the App Widgets documentation. Animation framework  * New ViewPropertyAnimator class   + A new ViewPropertyAnimator class provides a convenient way for developers to animate select properties onView objects. The class automaties and optimizes the animation of the properties and makes it easier to manage multiple simulataneous animations on a View object.   Using the ViewPropertyAnimator is straightforward. To animate properties for a View, call animate() to construct a ViewPropertyAnimator object for that View. Use the methods on the ViewPropertyAnimator to specify what property to animate and how to animate it. For example, to fade the View to transparent, callalpha(0);. The ViewPropertyAnimator object handles the details of configuring the underlying Animator class and starting it, then rendering the animation.   * Animation background color   + New getBackgroundColor() and setBackgroundColor(int) methods let you get/set the background color behind animations, for window animations only. Currently the background must be black, with any desired alpha level. * Getting animated fraction from ViewAnimator   + A new getAnimatedFraction() method lets you get the current animation fraction — the elapsed/interpolated fraction used in the most recent frame update — from a ValueAnimator.  UI framework  * Forced rendering of a layer   + A new buildLayer() method lets an application force a View's layer to be created and the View rendered into it immediately. For example, an application could use this method to render a View into its layer before starting an animation. If the View is complex, rendering it into the layer before starting the animation will avoid skipping frames. * Camera distance   + Applications can use a new method setCameraDistance(float) to set the distance from the camera to a View. This gives applications improved control over 3D transformations of the View, such as rotations. * Getting a calendar view from a DatePicker   + A new getCalendarView() method lets you get a CalendarView from a DatePicker instance. * Getting callbacks when views are detached   + A new View.OnAttachStateChangeListener lets you receive callbacks when a View is attached or detached from its window. Use addOnAttachStateChangeListener() to add a listener andaddOnAttachStateChangeListener() to remove it. * Fragment breadcrumb listener, new onInflate() signature   + A new method, setOnBreadCrumbClickListener(), provides a hook to let applications intercept fragment-breadcrumb clicks and take any action needed before going to the backstack entry or fragment that was clicked.   + In the Fragment class, onInflate(attrs, savedInstanceState) is deprecated. Please useonInflate(activity, attrs, savedInstanceState) instead. * Display search result in new tab   + An EXTRA\_NEW\_SEARCH data key for ACTION\_WEB\_SEARCH intents lets you open a search in a new browser tab, rather than in an existing one. * Drawable text cursor   + You can now specify a drawable to use as the text cursor using the new resource attribute textCursorDrawable. * Setting displayed child in remote views   + A new convenience method, setDisplayedChild(viewId, childIndex), is available in RemoteViewssubclasses, to let you set the child displayed in ViewAnimator and AdapterViewAnimator subclasses such asAdapterViewFlipper, StackView, ViewFlipper, and ViewSwitcher. * Generic keys for gamepads and other input devices   + KeyEvent adds a range of generic keycodes to accommodate gamepad buttons. The class also addsisGamepadButton(int) and several other helper methods for working with keycodes.  Graphics  * Helpers for managing bitmaps   + setHasAlpha(boolean) lets an app indicate that all of the pixels in a Bitmap are known to be opaque (false) or that some of the pixels may contain non-opaque alpha values (true). Note, for some configs (such as RGB\_565) this call is ignored, since it does not support per-pixel alpha values. This is meant as a drawing hint, as in some cases a bitmap that is known to be opaque can take a faster drawing case than one that may have non-opaque per-pixel alpha values.   + getByteCount() gets a Bitmap's size in bytes.   + getGenerationId() lets an application find out whether a Bitmap has been modified, such as for caching.   + sameAs(android.graphics.Bitmap) determines whether a given Bitmap differs from the current Bitmap, in dimension, configuration, or pixel data. * Setting camera location and rotation   + Camera adds two new methods rotate() and setLocation() for control of the camera's location, for 3D transformations.  Network  * High-performance Wi-Fi lock   + A new high-performance Wi-Fi lock lets applications maintain high-performance Wi-Fi connections even when the device screen is off. Applications that stream music, video, or voice for long periods can acquire the high-performance Wi-Fi lock to ensure streaming performance even when the screen is off. Because it uses more power, applications should acquire the high-performance Wi-Fi when there is a need for a long-running active connection.   To create a high-performance lock, pass WIFI\_MODE\_FULL\_HIGH\_PERF as the lock mode in a call tocreateWifiLock().   * More traffic stats   + Applications can now access statistics about more types of network usage using new methods in TrafficStats. Applications can use the methods to get UDP stats, packet count, TCP transmit/receive payload bytes and segments for a given UID. * SIP auth username   + Applications can now get and set the SIP auth username for a profile using the new methods getAuthUserName()and setAuthUserName().  Download Manager  * Handling of completed downloads   + Applications can now initiate downloads that notify users only on completion. To initiate this type of download, applications pass VISIBILITY\_VISIBLE\_NOTIFY\_ONLY\_COMPLETION in the setNotificationVisibility()method of the a request object.   + A new method, addCompletedDownload(), lets an application add a file to the downloads database, so that it can be managed by the Downloads application. * Show downloads sorted by size   + Applications can start the Downloads application in sort-by-size mode by adding the new extraINTENT\_EXTRAS\_SORT\_BY\_SIZE to an ACTION\_VIEW\_DOWNLOADS intent.  IME framework  * Getting an input method's extra value key   + The InputMethodSubtype adds the method containsExtraValueKey() to check whether an ExtraValue string is stored for the subtype and the method getExtraValueOf() to extract a specific key value from the ExtraValue hashmap.  Media  * New streaming audio formats   + The media framework adds built-in support for raw ADTS AAC content, for improved streaming audio, as well as support for FLAC audio, for highest quality (lossless) compressed audio content. See the Supported Media Formatsdocument for more information.    Launch controls on stopped applications Starting from Android 3.1, the system's package manager keeps track of applications that are in a stopped state and provides a means of controlling their launch from background processes and other applications.  Note that an application's stopped state is not the same as an Activity's stopped state. The system manages those two stopped states separately.  The platform defines two new intent flags that let a sender specify whether the Intent should be allowed to activate components in stopped application.   * FLAG\_INCLUDE\_STOPPED\_PACKAGES — Include intent filters of stopped applications in the list of potential targets to resolve against. * FLAG\_EXCLUDE\_STOPPED\_PACKAGES — Exclude intent filters of stopped applications from the list of potential targets.   When neither or both of these flags is defined in an intent, the default behavior is to include filters of stopped applications in the list of potential targets.  Note that the system adds FLAG\_EXCLUDE\_STOPPED\_PACKAGES to all broadcast intents. It does this to prevent broadcasts from background services from inadvertently or unnecessarily launching components of stoppped applications. A background service or application can override this behavior by adding the FLAG\_INCLUDE\_STOPPED\_PACKAGES flag to broadcast intents that should be allowed to activate stopped applications.  Applications are in a stopped state when they are first installed but are not yet launched and when they are manually stopped by the user (in Manage Applications). Notification of application first launch and upgrade The platform adds improved notification of application first launch and upgrades through two new intent actions:   * ACTION\_PACKAGE\_FIRST\_LAUNCH — Sent to the installer package of an application when that application is first launched (that is, the first time it is moved out of a stopped state). The data contains the name of the package. * ACTION\_MY\_PACKAGE\_REPLACED — Notifies an application that it was updated, with a new version was installed over an existing version. This is only sent to the application that was replaced. It does not contain any additional data. To receive it, declare an intent filter for this action. You can use the intent to trigger code that helps get your application back in proper running shape after an upgrade.   This intent is sent directly to the application, but only if the application was upgraded while it was in started state (not in a stopped state). Core utilities  * LRU cache   + A new LruCache class lets your applications benefit from efficient caching. Applications can use the class to reduce the time spent computing or downloading data from the network, while maintaining a sensible memory footprint for the cached data.LruCache is a cache that holds strong references to a limited number of values. Each time a value is accessed, it is moved to the head of a queue. When a value is added to a full cache, the value at the end of that queue is evicted and may become eligible for garbage collection. * File descriptor as int   + You can now get the native file descriptor int for a ParcelFileDescriptor using either of the new methodsgetFd() or detachFd().  WebKit  * File scheme cookies   + The CookieManager now supports cookies that use the file: URI scheme. You can usesetAcceptFileSchemeCookies() to enable/disable support for file scheme cookies, before constructing an instance of WebView or CookieManager. In a CookieManager instance, you can check whether file scheme cookies is enabled by calling allowFileSchemeCookies(). * Notification of login request   + To support the browser autologin features introduced in Android 3.0, the new methodonReceivedLoginRequest() notifies the host application that an autologin request for the user was processed. * Removed classes and interfaces   + Several classes and interfaces were removed from the public API, after previously being in deprecated state. See theAPI Differences Report for more information.    Browser The Browser application adds the following features to support web applications:   * Support for inline playback of video embedded in HTML5 <video> tag. Playback is hardware-accelerated where possible. * Layer support for fixed position elements for all sites (mobile and desktop).  New feature constants The platform adds new hardware feature constants that developers can declare in their application manifests, to inform external entities such as Google Play of the application's requirement for new hardware capabilities supported in this version of the platform. Developers declare these and other feature constants in <uses-feature> manifest elements.   * android.hardware.usb.accessory — The application uses the USB API to communicate with external hardware devices connected over USB and function as hosts. * android.hardware.usb.host — The application uses the USB API to communicate with external hardware devices connected over USB and function as devices.   Google Play filters applications based on features declared in <uses-feature> manifest elements. For more information about declaring features in an application manifest, read Google Play Filters. API Differences Report For a detailed view of all API changes in Android 3.1 (API Level 12), see the API Differences Report. API Level The Android 3.1 platform delivers an updated version of the framework API. The Android 3.1 API is assigned an integer identifier — **12** — 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 3.1 in your application, you need compile the application against the Android library that is provided in the Android 3.1 SDK platform. Depending on your needs, you might also need to add anandroid:minSdkVersion="12" attribute to the <uses-sdk> element in the application's manifest. | |  | | Android 3.0 APIs API Level: **11**  For developers, the Android 3.0 platform (HONEYCOMB) 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. The downloadable platform includes no external libraries.  For developers, the Android 3.0 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 3.0, use the Android SDK Manager to download the platform into your SDK. API Overview The sections below provide a technical overview of what's new for developers in Android 3.0, including new features and changes in the framework API since the previous version. Fragments A fragment is a new framework component that allows you to separate distinct elements of an activity into self-contained modules that define their own UI and lifecycle. To create a fragment, you must extend the Fragment class and implement several lifecycle callback methods, similar to an Activity. You can then combine multiple fragments in a single activity to build a multi-pane UI in which each pane manages its own lifecycle and user inputs.  You can also use a fragment without providing a UI and instead use the fragment as a worker for the activity, such as to manage the progress of a download that occurs only while the activity is running.  Additionally:   * Fragments are self-contained and you can reuse them in multiple activities * You can add, remove, replace and animate fragments inside the activity * You can add fragments to a back stack managed by the activity, preserving the state of fragments as they are changed and allowing the user to navigate backward through the different states * By providing alternative layouts, you can mix and match fragments, based on the screen size and orientation * Fragments have direct access to their container activity and can contribute items to the activity's Action Bar (discussed next)   To manage the fragments in your activity, you must use the FragmentManager, which provides several APIs for interacting with fragments, such as finding fragments in the activity and popping fragments off the back stack to restore their previous state.  To perform a transaction, such as add or remove a fragment, you must create a FragmentTransaction. You can then call methods such as add() remove(), or replace(). Once you've applied all the changes you want to perform for the transaction, you must call commit() and the system applies the fragment transaction to the activity.  For more information about using fragments, read the Fragments documentation. Several samples are also available in the API Demos application. Action Bar The Action Bar is a replacement for the traditional title bar at the top of the activity window. It includes the application logo in the left corner and provides a new interface for items in the Options Menu. Additionally, the Action Bar allows you to:   * Add menu items directly in the Action Bar—as "action items."   In your XML declaration for the menu item, include the android:showAsAction attribute with a value of "ifRoom". When there's enough room, the menu item appears directly in the Action Bar. Otherwise, the item is placed in the overflow menu, revealed by the menu icon on the right side of the Action Bar.   * Replace an action item with a widget (such as a search box)—creating an "action view."   In the XML declaration for the menu item, add the android:actionViewLayout attribute with a layout resource or theandroid:actionViewClass attribute with the class name of a widget. (You must also declare theandroid:showAsAction attribute so that the item appears in the Action Bar.) If there's not enough room in the Action Bar and the item appears in the overflow menu, it behaves like a regular menu item and does not show the widget.   * Add an action to the application logo and replace it with a custom logo   The application logo is automatically assigned the android.R.id.home ID, which the system delivers to your activity's onOptionsItemSelected() callback when touched. Simply respond to this ID in your callback method to perform an action such as go to your application's "home" activity.  To replace the icon with a logo, specify your application logo in the manifest file with the android:logo attribute, then call setDisplayUseLogoEnabled(true) in your activity.   * Add breadcrumbs to navigate backward through the back stack of fragments * Add tabs or a drop-down list to navigate through fragments * Customize the Action Bar with themes and backgrounds   The Action Bar is standard for all applications that use the new holographic theme, which is also standard when you set either the android:minSdkVersion or android:targetSdkVersion to "11".  For more information about the Action Bar, read the Action Bar documentation. Several samples are also available in theAPI Demos application. System clipboard Applications can now copy and paste data (beyond mere text) to and from the system-wide clipboard. Clipped data can be plain text, a URI, or an intent.  By providing the system access to the data you want the user to copy, through a content provider, the user can copy complex content (such as an image or data structure) from your application and paste it into another application that supports that type of content.  To start using the clipboard, get the global ClipboardManager object by callinggetSystemService(CLIPBOARD\_SERVICE).  To copy an item to the clipboard, you need to create a new ClipData object, which holds one or more ClipData.Itemobjects, each describing a single entity. To create a ClipData object containing just one ClipData.Item, you can use one of the helper methods, such as newPlainText(), newUri(), and newIntent(), which each return a ClipData object pre-loaded with the ClipData.Item you provide.  To add the ClipData to the clipboard, pass it to setPrimaryClip() for your instance of ClipboardManager.  You can then read a file from the clipboard (in order to paste it) by calling getPrimaryClip() on theClipboardManager. Handling the ClipData you receive can be complicated and you need to be sure you can actually handle the data type in the clipboard before attempting to paste it.  The clipboard holds only one piece of clipped data (a ClipData object) at a time, but one ClipData can contain multipleClipData.Items.  For more information, read the Copy and Paste documentation. You can also see a simple implementation of copy and paste in the API Demos sample and a more complete implementation in the Note Pad sample. Drag and drop New APIs simplify drag and drop operations in your application's user interface. A drag operation is the transfer of some kind of data—carried in a ClipData object—from one place to another. The start and end point for the drag operation is aView, so the APIs that directly handle the drag and drop operations are in the View class.  A drag and drop operation has a lifecycle that's defined by several drag actions—each defined by a DragEvent object—such as ACTION\_DRAG\_STARTED, ACTION\_DRAG\_ENTERED, and ACTION\_DROP. Each view that wants to participate in a drag operation can listen for these actions.  To begin dragging content in your activity, call startDrag() on a View, providing a ClipData object that represents the data to drag, a View.DragShadowBuilder to facilitate the "shadow" that users see under their fingers while dragging, and an Object that can share information about the drag object with views that may receive the object.  To accept a drag object in a View (receive the "drop"), register the view with an OnDragListener by callingsetOnDragListener(). When a drag event occurs on the view, the system calls onDrag() for the OnDragListener, which receives a DragEvent describing the type of drag action has occurred (such as ACTION\_DRAG\_STARTED,ACTION\_DRAG\_ENTERED, and ACTION\_DROP). During a drag, the system repeatedly calls onDrag() for the view underneath the drag, to deliver a stream of drag events. The receiving view can inquire the event type delivered toonDragEvent() by calling getAction() on the DragEvent.  **Note:** Although a drag event may carry a ClipData object, this is not related to the system clipboard. A drag and drop operation should never put the dragged data in the system clipboard.  For more information, read the Dragging and Dropping documentation. You can also see an implementation of drag and drop in the API Demos application and the Honeycomb Gallery application. App widgets Android 3.0 supports several new widget classes for more interactive app widgets on the users Home screen, including:GridView, ListView, StackView, ViewFlipper, and AdapterViewFlipper.  More importantly, you can use the new RemoteViewsService to create app widgets with collections, using widgets such as GridView, ListView, and StackView that are backed by remote data, such as from a content provider.  The AppWidgetProviderInfo class (defined in XML with an <appwidget-provider> element) also supports two new fields: autoAdvanceViewId and previewImage. The autoAdvanceViewId field lets you specify the view ID of the app widget subview that should be auto-advanced by the app widget’s host. The previewImage field specifies a preview of what the app widget looks like and is shown to the user from the widget picker. If this field is not supplied, the app widget's icon is used for the preview.  To help create a preview image for your app widget (to specify in the previewImage field), the Android emulator includes an application called "Widget Preview." To create a preview image, launch this application, select the app widget for your application and set it up how you'd like your preview image to appear, then save it and place it in your application's drawable resources.  You can see an implementation of the new app widget features in the StackView App Widget and Weather List Widgetapplications. Status bar notifications The Notification APIs have been extended to support more content-rich status bar notifications, plus a newNotification.Builder class allows you to easily create Notification objects.  New features include:   * Support for a large icon in the notification, using setLargeIcon(). This is usually for social applications to show the contact photo of the person who is the source of the notification or for media apps to show an album thumbnail. * Support for custom layouts in the status bar ticker, using setTicker(). * Support for custom notification layouts to include buttons with PendingIntents, for more interactive notification widgets. For example, a notification can control music playback without starting an activity.  Content loaders New framework APIs facilitate asynchronous loading of data using the Loader class. You can use it in combination with UI components such as views and fragments to dynamically load data from worker threads. The CursorLoader subclass is specially designed to help you do so for data backed by a ContentProvider.  All you need to do is implement the LoaderCallbacks interface to receive callbacks when a new loader is requested or the data has changed, then call initLoader() to initialize the loader for your activity or fragment.  For more information, read the Loaders documentation. You can also see example code using loaders in the LoaderCursorand LoaderThrottle samples. Bluetooth A2DP and headset APIs Android now includes APIs for applications to verify the state of connected Bluetooth A2DP and headset profile devices. For example, applications can identify when a Bluetooth headset is connected for listening to music and notify the user as appropriate. Applications can also receive broadcasts for vendor specific AT commands and notify the user about the state of the connected device, such as when the connected device's battery is low.  You can initialize the respective BluetoothProfile by calling getProfileProxy() with either the A2DP or HEADSETprofile constant and a BluetoothProfile.ServiceListener to receive callbacks when the Bluetooth client is connected or disconnected. Animation framework An all new flexible animation framework allows you to animate arbitrary properties of any object (View, Drawable, Fragment, Object, or anything else). It allows you to define several aspects of an animation, such as:   * Duration * Repeat amount and behavior * Type of time interpolation * Animator sets to play animations together, sequentially, or after specified delays * Frame refresh delay   You can define these animation aspects, and others, for an object's int, float, and hexadecimal color values, by default. That is, when an object has a property field for one of these types, you can change its value over time to affect an animation. To animate any other type of value, you tell the system how to calculate the values for that given type, by implementing the TypeEvaluator interface.  There are two animators you can use to animate the values of a property: ValueAnimator and ObjectAnimator. TheValueAnimator computes the animation values, but is not aware of the specific object or property that is animated as a result. It simply performs the calculations, and you must listen for the updates and process the data with your own logic. The ObjectAnimator is a subclass of ValueAnimator and allows you to set the object and property to animate, and it handles all animation work. That is, you give the ObjectAnimator the object to animate, the property of the object to change over time, and a set of values to apply to the property over time, then start the animation.  Additionally, the LayoutTransition class enables automatic transition animations for changes you make to your activity layout. To enable transitions for part of the layout, create a LayoutTransition object and set it on anyViewGroup by calling setLayoutTransition(). This causes default animations to run whenever items are added to or removed from the group. To specify custom animations, call setAnimator() on the LayoutTransition and provide a custom Animator, such as a ValueAnimator or ObjectAnimator discussed above.  For more information, see the Property Animation documentation. You can also see several samples using the animation APIs in the API Demos application. Extended UI framework  * **Multiple-choice selection for ListView and GridView**   New CHOICE\_MODE\_MULTIPLE\_MODAL mode for setChoiceMode() allows users to select multiple items from aListView or GridView. When used in conjunction with the Action Bar, users can select multiple items and then select the action to perform from a list of options in the Action Bar (which has transformed into a Multi-choice Action Mode).  To enable multiple-choice selection, call setChoiceMode(CHOICE\_MODE\_MULTIPLE\_MODAL) and register aMultiChoiceModeListener with setMultiChoiceModeListener().  When the user performs a long-press on an item, the Action Bar switches to the Multi-choice Action Mode. The system notifies the MultiChoiceModeListener when items are selected by calling onItemCheckedStateChanged().  For an example of multiple-choice selection, see the List15. java class in the API Demos sample application.   * **New APIs to transform views**   New APIs allow you to easily apply 2D and 3D transformations to views in your activity layout. New transformations are made possible with a set of object properties that define the view's layout position, orientation, transparency and more.  New methods to set the view properties include: setAlpha(), setBottom(), setLeft(), setRight(), setBottom(),setPivotX(), setPivotY(), setRotationX(), setRotationY(), setScaleX(), setScaleY(), setAlpha(), and others.  Some methods also have a corresponding XML attribute that you can specify in your layout file, to apply a default transformation. Available attributes include: translationX, translationY, rotation, rotationX, rotationY,scaleX, scaleY, transformPivotX, transformPivotY, and alpha.  Using some of these new view properties in combination with the new animation framework (discussed above), you can easily apply some fancy animations to your views. For example, to rotate a view on its y-axis, supplyObjectAnimator with the View, the "rotationY" property, and the start and end values:  ObjectAnimator animator = ObjectAnimator.ofFloat(myView, "rotationY", 0, 360);  animator.setDuration(2000);  animator.start();   * **New holographic themes**   The standard system widgets and overall look have been redesigned and incorporate a new "holographic" user interface theme. The system applies the new theme using the standard style and theme system.  Any application that targets the Android 3.0 platform—by setting either the android:minSdkVersion orandroid:targetSdkVersion value to "11"—inherits the holographic theme by default. However, if your application also applies its own theme, then your theme will override the holographic theme, unless you update your styles to inherit the holographic theme.  To apply the holographic theme to individual activities or to inherit them in your own theme definitions, use one of several new Theme.Holo themes. If your application is compatible with version of Android lower than 3.0 and applies custom themes, then you should select a theme based on platform version.   * **New widgets**   + AdapterViewAnimator   Base class for an AdapterView that performs animations when switching between its views.   * + AdapterViewFlipper   Simple ViewAnimator that animates between two or more views that have been added to it. Only one child is shown at a time. If requested, it can automatically flip between each child at a regular interval.   * + CalendarView   Allows users to select dates from a calendar by touching the date and can scroll or fling the calendar to a desired date. You can configure the range of dates available in the widget.   * + ListPopupWindow   Anchors itself to a host view and displays a list of choices, such as for a list of suggestions when typing into anEditText view.   * + NumberPicker   Enables the user to select a number from a predefined range. The widget presents an input field and up and down buttons for selecting a number. Touching the input field allows the user to scroll through values or touch again to directly edit the current value. It also allows you to map positions to strings, so that the corresponding string is displayed instead of the index position.   * + PopupMenu   Displays a Menu in a modal popup window that's anchored to a view. The popup appears below the anchor view if there is room, or above it if there is not. If the IME (soft keyboard) is visible, the popup does not overlap the IME it until the user touches the menu.   * + SearchView   Provides a search box that you can configure to deliver search queries to a specified activity and display search suggestions (in the same manner as the traditional search dialog). This widget is particularly useful for offering a search widget in the Action Bar. For more information, see Creating a Search Interface.   * + StackView   A view that displays its children in a 3D stack and allows users to swipe through views like a rolodex. Graphics  * **Hardware accelerated 2D graphics**   You can now enable the OpenGL renderer for your application by setting android:hardwareAccelerated="true" in your manifest element's <application> element or for individual <activity> elements.  This flag helps applications by making them draw faster. This results in smoother animations, smoother scrolling, and overall better performance and response to user interaction.   * **View support for hardware and software layers**   By default, a View has no layer specified. You can specify that the view be backed by either a hardware or software layer, specified by values LAYER\_TYPE\_HARDWARE and LAYER\_TYPE\_SOFTWARE, using setLayerType() or thelayerType attribute.  A hardware layer is backed by a hardware specific texture (generally Frame Buffer Objects or FBO on OpenGL hardware) and causes the view to be rendered using Android's hardware rendering pipeline, but only if hardware acceleration is turned on for the view hierarchy. When hardware acceleration is turned off, hardware layers behave exactly as software layers.  A software layer is backed by a bitmap and causes the view to be rendered using Android's software rendering pipeline, even if hardware acceleration is enabled. Software layers should be avoided when the affected view tree updates often. Every update will require to re-render the software layer, which can potentially be slow.  For more information, see the LAYER\_TYPE\_HARDWARE and LAYER\_TYPE\_SOFTWARE documentation.   * **Renderscript 3D graphics engine**   Renderscript is a runtime 3D framework that provides both an API for building 3D scenes as well as a special, platform-independent shader language for maximum performance. Using Renderscript, you can accelerate graphics operations and data processing. Renderscript is an ideal way to create high-performance 3D effects for applications, wallpapers, carousels, and more.  For more information, see the 3D Rendering and Computation with Renderscript documentation. Media  * **Time lapse video**   Camcorder APIs now support the ability to record time lapse video. The setCaptureRate() sets the rate at which frames should be captured.   * **Texture support for image streams**   New SurfaceTexture allows you to capture an image stream as an OpenGL ES texture. By callingsetPreviewTexture() for your Camera instance, you can specify the SurfaceTexture upon which to draw video playback or preview frames from the camera.   * **HTTP Live streaming**   Applications can now pass an M3U playlist URL to the media framework to begin an HTTP Live streaming session. The media framework supports most of the HTTP Live streaming specification, including adaptive bit rate. See theSupported Media Formats document for more information.   * **EXIF data**   The ExifInterface includes new fields for photo aperture, ISO, and exposure time.   * **Camcorder profiles**   New hasProfile() method and several video quality profiles (such as QUALITY\_1080P, QUALITY\_720P,QUALITY\_CIF, and others) allow you to determine camcorder quality options.   * **Digital media file transfer**   The platform includes built-in support for Media/Picture Transfer Protocol (MTP/PTP) over USB, which lets users easily transfer any type of media files between devices and to a host computer. Developers can build on this support, creating applications that let users create or manage rich media files that they may want to transfer or share across devices.   * **Digital rights management (DRM)**   New extensible digital rights management (DRM) framework for checking and enforcing digital rights. It's implemented in two architectural layers:   * + A DRM framework API, which is exposed to applications and runs through the Dalvik VM for standard applications.   + A native code DRM manager that implements the framework API and exposes an interface for DRM plug-ins to handle rights management and decryption for various DRM schemes.   For application developers, the framework offers an abstract, unified API that simplifies the management of protected content. The API hides the complexity of DRM operations and allows a consistent operation mode for both protected and unprotected content, and across a variety of DRM schemes.  For device manufacturers, content owners, and Internet digital media providers the DRM framework?s plugin API provides a means of adding support for a DRM scheme of choice into the Android system, for secure enforcement of content protection.  The preview release does not provide any native DRM plug-ins for checking and enforcing digital rights. However, device manufacturers may ship DRM plug-ins with their devices.  You can find all of the DRM APIs in the android.drm package. Keyboard support  * Support for Control, Meta, Caps Lock, Num Lock and Scroll Lock modifiers. For more information, see META\_CTRL\_ONand related fields. * Support for full desktop-style keyboards, including support for keys such as Escape, Home, End, Delete and others. You can determine whether key events are coming from a full keyboard by querying getKeyboardType() and checking forKeyCharacterMap.FULL * TextView now supports keyboard-based cut, copy, paste, and select-all, using the key combinations Ctrl+X, Ctrl+C, Ctrl+V, and Ctrl+A. It also supports PageUp/PageDown, Home/End, and keyboard-based text selection. * KeyEvent adds several new methods to make it easier to check the key modifier state correctly and consistently. SeehasModifiers(int), hasNoModifiers(), metaStateHasModifiers(), metaStateHasNoModifiers(). * Applications can implement custom keyboard shortcuts by subclassing Activity, Dialog, or View and implementingonKeyShortcut(). The framework calls this method whenever a key is combined with Ctrl key. When creating anOptions Menu, you can register keyboard shortcuts by setting either the android:alphabeticShortcut orandroid:numericShortcut attribute for each <item> element (or with setShortcut()). * Android 3.0 includes a new "virtual keyboard" device with the id KeyCharacterMap.VIRTUAL\_KEYBOARD. The virtual keyboard has a desktop-style US key map which is useful for synthesizing key events for testing input.  Split touch events Previously, only a single view could accept touch events at one time. Android 3.0 adds support for splitting touch events across views and even windows, so different views can accept simultaneous touch events.  Split touch events is enabled by default when an application targets Android 3.0. That is, when the application has set either the android:minSdkVersion or android:targetSdkVersion attribute's value to "11".  However, the following properties allow you to disable split touch events across views inside specific view groups and across windows.   * The android:splitMotionEvents attribute for view groups allows you to disable split touch events that occur between child views in a layout. For example: * <LinearLayout android:splitMotionEvents="false" ... > * ...   </LinearLayout>  This way, child views in the linear layout cannot split touch events—only one view can receive touch events at a time.   * The android:windowEnableSplitTouch style property allows you to disable split touch events across windows, by applying it to a theme for the activity or entire application. For example: * <style name="NoSplitMotionEvents" parent="android:Theme.Holo"> * <item name="android:windowEnableSplitTouch">false</item> * ...   </style>  When this theme is applied to an <activity> or <application>, only touch events within the current activity window are accepted. For example, by disabling split touch events across windows, the system bar cannot receive touch events at the same time as the activity. This does not affect whether views inside the activity can split touch events—by default, the activity can still split touch events across views.  For more information about creating a theme, read Applying Styles and Themes. WebKit  * New WebViewFragment class to create a fragment composed of a WebView. * New WebSettings methods:   + setDisplayZoomControls() allows you to hide the on-screen zoom controls while still allowing the user to zoom with finger gestures (setBuiltInZoomControls() must be set true).   + New WebSettings method, setEnableSmoothTransition(), allows you to enable smooth transitions when panning and zooming. When enabled, WebView will choose a solution to maximize the performance (for example, the WebView's content may not update during the transition). * New WebView methods:   + onPause() callback, to pause any processing associated with the WebView when it becomes hidden. This is useful to reduce unnecessary CPU or network traffic when the WebView is not in the foreground.   + onResume() callback, to resume processing associated with the WebView, which was paused during onPause().   + saveWebArchive() allows you to save the current view as a web archive on the device.   + showFindDialog() initiates a text search in the current view.  Browser The Browser application adds the following features to support web applications:   * **Media capture**   As defined by the HTML Media Capture specification, the Browser allows web applications to access audio, image and video capture capabilities of the device. For example, the following HTML provides an input for the user to capture a photo to upload:  <input type="file" accept="image/\*;capture=camera" />  Or by excluding the capture=camera parameter, the user can choose to either capture a new image with the camera or select one from the device (such as from the Gallery application).   * **Device Orientation**   As defined by the Device Orientation Event specification, the Browser allows web applications to listen to DOM events that provide information about the physical orientation and motion of the device.  The device orientation is expressed with the x, y, and z axes, in degrees and motion is expressed with acceleration and rotation rate data. A web page can register for orientation events by calling window.addEventListener with event type "deviceorientation" and register for motion events by registering the "devicemotion" event type.   * **CSS 3D Transforms**   As defined by the CSS 3D Transform Module specification, the Browser allows elements rendered by CSS to be transformed in three dimensions. JSON utilities New classes, JsonReader and JsonWriter, help you read and write JSON streams. The new APIs complement theorg.json classes, which manipulate a document in memory.  You can create an instance of JsonReader by calling its constructor method and passing the InputStreamReader that feeds the JSON string. Then begin reading an object by calling beginObject(), read a key name with nextName(), read the value using methods respective to the type, such as nextString() and nextInt(), and continue doing so whilehasNext() is true.  You can create an instance of JsonWriter by calling its constructor and passing the appropriate OutputStreamWriter. Then write the JSON data in a manner similar to the reader, using name() to add a property name and an appropriatevalue() method to add the respective value.  These classes are strict by default. The setLenient() method in each class configures them to be more liberal in what they accept. This lenient parse mode is also compatible with the org.json's default parser. New feature constants The <uses-feature> manfest element should be used to inform external entities (such as Google Play) of the set of hardware and software features on which your application depends. In this release, Android adds the following new constants that applications can declare with this element:   * "android.hardware.faketouch"   When declared, this indicates that the application is compatible with a device that offers an emulated touchscreen (or better). A device that offers an emulated touchscreen provides a user input system that can emulate a subset of touchscreen capabilities. An example of such an input system is a mouse or remote control that drives an on-screen cursor. Such input systems support basic touch events like click down, click up, and drag. However, more complicated input types (such as gestures, flings, etc.) may be more difficult or impossible on faketouch devices (and multitouch gestures are definitely not possible).  If your application does not require complicated gestures and you do not want your application filtered from devices with an emulated touchscreen, you should declare "android.hardware.faketouch" with a <uses-feature>element. This way, your application will be available to the greatest number of device types, including those that provide only an emulated touchscreen input.  All devices that include a touchscreen also support "android.hardware.faketouch", because touchscreen capabilities are a superset of faketouch capabilities. Thus, unless you actually require a touchscreen, you should add a<uses-feature> element for faketouch. New permissions  * "android.permission.BIND\_REMOTEVIEWS"   This must be declared as a required permission in the <service> manifest element for an implementation ofRemoteViewsService. For example, when creating an App Widget that uses RemoteViewsService to populate a collection view, the manifest entry may look like this:  <service android:name=".widget.WidgetService"      android:exported="false"      android:permission="android.permission.BIND\_REMOTEVIEWS" /> New platform technologies  * **Storage**   + ext4 file system support to enable onboard eMMC storage.   + FUSE file system to support MTP devices.   + USB host mode support to support keyboards and USB hubs.   + Support for MTP/PTP * **Linux Kernel**   + Upgraded to 2.6.36 * **Dalvik VM**   + New code to support and optimize for SMP   + Various improvements to the JIT infrastructure   + Garbage collector improvements:     - Tuned for SMP     - Support for larger heap sizes     - Unified handling for bitmaps and byte buffers * **Dalvik Core Libraries**   + New, much faster implementation of NIO (modern I/O library)   + Improved exception messages   + Correctness and performance fixes throughout  API differences report For a detailed view of all API changes in Android 3.0 (API Level 11), see the API Differences Report. API Level The Android 3.0 platform delivers an updated version of the framework API. The Android 3.0 API is assigned an integer identifier — **11** — 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 3.0 in your application, you need compile the application against the Android library that is provided in the Android 3.0 SDK platform. Depending on your needs, you might also need to add anandroid:minSdkVersion="11" attribute to the <uses-sdk> element in the application's manifest. If your application is designed to run only on Android 2.3 and higher, declaring the attribute prevents the application from being installed on earlier versions of the platform. | | | |  | | |  Gingerbread The Android 2.3 platform introduces many new and exciting features for users and developers. This document provides a glimpse at some of the new features and technologies in Android 2.3. For detailed information about the new developer APIs, see the Android 2.3 version notes.   * New User Features * New Developer Features * New Platform Technologies  New User Features http://developer.android.com/sdk/images/2.3/home-menu.pnghttp://developer.android.com/sdk/images/2.3/home-plain.png UI refinements for simplicity and speed The user interface is refined in many ways across the system, making it easier to learn, faster to use, and more power-efficient. A simplified visual theme of colors against black brings vividness and contrast to the notification bar, menus, and other parts of the UI. Changes in menus and settings make it easier for the user to navigate and control the features of the system and device. Faster, more intuitive text input The Android soft keyboard is redesigned and optimized for faster text input and editing. The keys themselves are reshaped and repositioned for improved targeting, making them easier to see and press accurately, even at high speeds. The keyboard also displays the current character and dictionary suggestions in a larger, more vivid style that is easier to read.  The keyboard adds the capability to correct entered words from suggestions in the dictionary. As the user selects a word already entered, the keyboard displays suggestions that the user can choose from, to replace the selection. The user can also switch to voice input mode to replace the selection. Smart suggestions let the user accept a suggestion and then return to correct it later, if needed, from the original set of suggestions.  New multitouch key-chording lets the user quickly enter numbers and symbols by pressing Shift+<letter> and ?123+<symbol>, without needing to manually switch input modes. From certain keys, users can also access a popup menu of accented characters, numbers, and symbols by holding the key and sliding to select a character.  http://developer.android.com/sdk/images/2.3/onetouch.png  http://developer.android.com/sdk/images/2.3/selection.png One-touch word selection and copy/paste When entering text or viewing a web page, the user can quickly select a word by press-hold, then copy to the clipboard and paste. Pressing on a word enters a free-selection mode — the user can adjust the selection area as needed by dragging a set of bounding arrows to new positions, then copy the bounded area by pressing anywhere in the selection area. For text entry, the user can slide-press to enter a cursor mode, then reposition the cursor easily and accurately by dragging the cursor arrow. With both the selection and cursor modes, no use of a trackball is needed.  http://developer.android.com/sdk/images/2.3/running.png  http://developer.android.com/sdk/images/2.3/power.png Improved power management The Android system takes a more active role in managing apps that are keeping the device awake for too long or that are consuming CPU while running in the background. By managing such apps — closing them if appropriate — the system helps ensure best possible performance and maximum battery life.  The system also gives the user more visibility over the power being consumed by system components and running apps. The Application settings provides an accurate overview of how the battery is being used, with details of the usage and relative power consumed by each component or application. Control over applications A shortcut to the Manage Applications control now appears in the Options Menu in the Home screen and Launcher, making it much easier to check and manage application activity. Once the user enters Manage Applications, a new Running tab displays a list of active applications and the storage and memory being used by each. The user can read further details about each application and if necessary stop an application or report feedback to its developer. New ways of communicating, organizing An updated set of standard applications lets the user take new approaches to managing information and relationships.  http://developer.android.com/sdk/images/2.3/sipcall.png http://developer.android.com/sdk/images/2.3/ffc.png    **Internet calling**  The user can make voice calls over the internet to other users who have SIP accounts. The user can add an internet calling number (a SIP address) to any Contact and can initiate a call from Quick Contact or Dialer. To use internet calling, the user must create an account at the SIP provider of their choice — SIP accounts are not provided as part of the internet calling feature. Additionally, support for the platform's SIP and internet calling features on specific devices is determined by their manufacturers and associated carriers.  http://developer.android.com/sdk/images/2.3/nfc.png  **Near-field communications**  An NFC Reader application lets the user read and interact with near-field communication (NFC) tags. For example, the user can “touch” or “swipe” an NFC tag that might be embedded in a poster, sticker, or advertisement, then act on the data read from the tag. A typical use would be to read a tag at a restaurant, store, or event and then rate or register by jumping to a web site whose URL is included in the tag data. NFC communication relies on wireless technology in the device hardware, so support for the platform's NFC features on specific devices is determined by their manufacturers.  **Downloads management**  The Downloads application gives the user easy access to any file downloaded from the browser, email, or another application. Downloads is built on an completely new download manager facility in the system that any other applications can use, to more easily manage and store their downloads.  **Camera**  The application now lets the user access multiple cameras on the device, including a front-facing camera, if available. New Developer Features Android 2.3 delivers a variety of features and APIs that let developers bring new types of applications to the Android platform.   * Enhancements for gaming * New forms of communication * Rich multimedia  Enhancements for gaming **Performance**  Android 2.3 includes a variety of improvements across the system that make common operations faster and more efficient for all applications. Of particular interest to game developers are:   * Concurrent garbage collector — The Dalivik VM introduces a new, concurrent garbage collector that minimizes application pauses, helping to ensure smoother animation and increased responsiveness in games and similar applications. * Faster event distribution — The plaform now handles touch and keyboard events faster and more efficiently, minimizing CPU utilization during event distribution. The changes improve responsiveness for all applications, but especially benefit games that use touch events in combination with 3D graphics or other CPU-intensive operations. * Updated video drivers — The platform uses updated third-party video drivers that improve the efficiency of OpenGL ES operations, for faster overall 3D graphics performance.   **Native input and sensor events**  Applications that use native code can now receive and process input and sensor events directly in their native code, which dramatically improves efficiency and responsiveness.  Native libraries exposed by the platform let applications handle the same types of input events as those available through the framework. Applications can receive events from all supported sensor types and can enable/disable specific sensors and manage event delivery rate and queueing.  **Gyroscope and other new sensors, for improved 3D motion processing**  Android 2.3 adds API support for several new sensor types, including gyroscope, rotation vector, linear acceleration, gravity, and barometer sensors. Applications can use the new sensors in combination with any other sensors available on the device, to track three-dimensional device motion and orientation change with high precision and accuracy. For example, a game application could use readings from a gyroscope and accelerometer on the device to recognize complex user gestures and motions, such as tilt, spin, thrust, and slice.  **Open API for native audio**  The platform provides a software implementation of Khronos OpenSL ES, a standard API that gives applications access to powerful audio controls and effects from native code. Applications can use the API to manage audio devices and control audio input, output, and processing directly from native code.  **Native graphics management**  The platform provides an interface to its Khronos EGL library, which lets applications manage graphics contexts and create and manage OpenGL ES textures and surfaces from native code.  **Native access to Activity lifecycle, window management**  Native applications can declare a new type of Activity class, NativeActivity whose lifecycle callbacks are implemented directly in native code. The NativeActivity and its underlying native code run in the system just as do other Activities — they run in the application's system process and execute on the application's main UI thread, and they receive the same lifecycle callbacks as do other Activities.  The platform also exposes native APIs for managing windows, including the ability to lock/unlock the pixel buffer to draw directly into it. Through the API, applications can obtain a native window object associated with a framework Surface object and interact with it directly in native code.  **Native access to assets, storage**  Applications can now access a native Asset Manager API to retrieve application assets directly from native code without needing to go through JNI. If the assets are compressed, the platform does streaming decompression as the application reads the asset data. There is no longer a limit on the size of compressed .apk assets that can be read.  Additionally, applications can access a native Storage Manager API to work directly with OBB files downloaded and managed by the system. Note that although platform support for OBB is available in Android 2.3, development tools for creating and managing OBB files will not be available until early 2011.  **Robust native development environment**  The Android NDK (r5 or higher) provides a complete set of tools, toolchains, and libraries for developing applications that use the rich native environment offered by the Android 2.3 platform. For more information or to download the NDK, please see the Android NDK page. New forms of communication **Internet telephony**  Developers can now add SIP-based internet telephony features to their applications. Android 2.3 includes a full SIP protocol stack and integrated call management services that let applications easily set up outgoing and incoming voice calls, without having to manage sessions, transport-level communication, or audio record or playback directly.  Support for the platform's SIP and internet calling features on specific devices is determined by their manufacturers and associated carriers.  **Near Field Communications (NFC)**  The platform's support for Near Field Communications (NFC) lets developers get started creating a whole new class of applications for Android. Developers can create new applications that offer proximity-based information and services to users, organizations, merchants, and advertisers.  Using the NFC API, applications can read and respond to NFC tags “discovered” as the user “touches” an NFC-enabled device to elements embedded in stickers, smart posters, and even other devices. When a tag of interest is collected, applications can respond to the tag, read messages from it, and then store the messages, prompting the user as needed.  Starting from Android 2.3.3, applications can also write to tags and set up peer-to-peer connections with other NFC devices.  NFC communication relies on wireless technology in the device hardware, so support for the platform's NFC features on specific devices is determined by their manufacturers. Rich multimedia **Mixable audio effects**  A new audio effects API lets developers easily create rich audio environments by adding equalization, bass boost, headphone virtualization (widened soundstage), and reverb to audio tracks and sounds. Developers can mix multiple audio effects in a local track or apply effects globally, across multiple tracks.  **Support for new media formats**  The platform now offers built-in support for the VP8 open video compression format and the WebM open container format. The platform also adds support for AAC encoding and AMR wideband encoding (in software), so that applications can capture higher quality audio than narrowband.  **Access to multiple cameras**  The Camera API now lets developers access any cameras that are available on a device, including a front-facing camera. Applications can query the platform for the number of cameras on the device and their types and characteristics, then open the camera needed. For example, a video chat application might want to access a front-facing camera that offers lower-resolution, while a photo application might prefer a back-facing camera that offers higher-resolution. New Platform TechnologiesMedia Framework  * New media framework fully replaces OpenCore, maintaining all previous codec/container support for encoding and decoding. * Integrated support for the VP8 open video compression format and the WebM open container format * Adds AAC encoding and AMR wideband encoding  Linux Kernel  * Upgraded to 2.6.35  Networking  * SIP stack, configurable by device manufacturer * Support for Near Field Communications (NFC), configurable by device manufacturer * Updated BlueZ stack  Dalvik runtime  * Dalvik VM:   + Concurrent garbage collector (target sub-3ms pauses)   + Adds further JIT (code-generation) optimizations   + Improved code verification   + StrictMode debugging, for identifying performance and memory issues * Core libraries:   + Expanded I18N support (full worldwide encodings, more locales)   + Faster Formatter and number formatting. For example, float formatting is 2.5x faster.   + HTTP responses are gzipped by default. XML and JSON API response sizes may be reduced by 60% or more.   + New collections and utilities APIs   + Improved network APIs   + Improved file read and write controls   + Updated JDBC * Updates from upstream projects:   + OpenSSL 1.0.0a   + BouncyCastle 1.45   + ICU 4.4   + zlib 1.2.5  |  | | --- | | **Android 2.3.4 APIs**  *API Level:* **10**  Android 2.3.4 (GINGERBREAD\_MR1) is a maintenance release that adds several bug fixes and patches to the Android 2.3 platform, without any API changes from Android 2.3.3. Additionally, Android 2.3.4 brings support for the Open Accessory API to mobile devices, through the optional Open Accessory Library.  For developers, the Android 2.3.4 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 2.3.4, use the Android SDK Manager to download the platform into your SDK.  **API Overview**  Android 2.3.4 provides the same framework API to applications as Android 2.3.3 (API level 10). For a summary of the API, see the Android 2.3.3 version notes.  **Open Accessory Library**  *Open Accessory* is a new capability for integrating connected peripherals with applications running on the platform. The capability is based on a USB (Universal Serial Bus) stack built into the platform and an API exposed to applications. Peripherals that attach to Android-powered devices as accessories connect as USB hosts.  Open Accessory is introduced in Android 3.1 (API level 12), but is made available to devices running Android 2.3.4 by means of an optional external library, the Open Accessory Library. The library exposes a framework API that lets applications discover, communicate with, and manage a variety of device types connected over USB. It also provides the implementation of the API against parts of the Android platform that are not directly exposed to applications in Android 2.3.4.  The Open Accessory Library is optional on any given device. Device manufacturers may choose whether to include the Open Accessory Library in their products or exclude it. The library is forward-compatible with Android 3.1, so applications developed against Android 2.3.4 will run properly on devices running Android 3.1, if those devices support USB accessories.  The API provided by the Open Accessory Library is based on the Open Accessory API provided in Android 3.1. In most areas, you can use the same techniques and APIs. However, developing for the Open Accessory Library on Android 2.3.4 differs from the standard USB API in these ways:   * Obtaining a UsbManager object — To obtain a UsbManager object when using the add-on library, use the helper method getInstance() rather than getSystemService() For example:   UsbManager manager = UsbManager.getInstance(this);   * Obtaining a UsbAccessory from a filtered intent — When you filter for a connected device or accessory with an intent filter, the UsbAccessory object is contained inside the intent that is passed to your application. If you are using the add-on library, you can get the UsbAccessory object in the following manner:   UsbAccessory accessory = UsbManager.getAccessory(intent)   * No USB host support — Android 2.3.4 and the Open Accessory Library do not support USB host mode (for example, through UsbDevice), although USB host mode is supported in Android 3.1. An Android-powered device running Android 2.3.4 can not function as a USB host. The library enables the Android-powered device to function as a peripheral only, with the connected accessory functioning as USB host (through UsbAccessory).   To develop apps using the Open Accessory Library, you need:   * The latest version of the Android SDK tools * The latest version of the Google APIs add-on, which includes the library itself (for linking) * An actual hardware device running Android 2.3.4 (or Android 3.1) with USB accessories support, for runtime testing against connected devices   For a full discussion of how to develop applications that interact with USB accessories, please see the related developer documentation.  Additionally, developers can request filtering on Google Play, such that their applications are not available to users whose devices do not provide the appropriate accessory support. To request filtering, add the element below to the application manifest:  <uses-feature    android:name="android.hardware.usb.accessory"    android:required="true">  **API Level**  The Android 2.3.4 platform does *not* increment the API level — it uses the same API level as Android 2.3.3, API level 10.  To use APIs introduced in API level 10 in your application, you need compile the application against the Android library that is provided in the latest version of the Google APIs Add-On, which also includes the Open Accessory Library.  Depending on your needs, you might also need to add an android:minSdkVersion="10" attribute to the <uses-sdk>element in the application's manifest. If your application is designed to run only on Android 2.3.3 and higher, declaring the attribute prevents the application from being installed on earlier versions of the platform. | |  | | Android 2.3.3 APIs API Level: **10**  Android 2.3.3 (GINGERBREAD\_MR1) is a small feature release that adds several improvements and APIs to the Android 2.3 platform.  For developers, the Android 2.3.3 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 2.3.3, use the Android SDK Manager to download the platform into your SDK. API Overview The sections below provide a technical overview of what's new for developers in 2.3.3, including new features and changes in the framework API since the previous version. Near Field Communications (NFC) Android 2.3.3 provides improved and extended support for NFC, to allow applications to interact with more types of tags in new ways.  A new, comprehensive set of APIs give applications read and write access to a wider range of standard tag technologies, including:   * NFC-A (ISO 14443-3A) * NFC-B (ISO 14443-3B) * NFC-F (JIS 6319-4) * NFC-V (ISO 15693) * ISO-DEP (ISO 14443-4) * MIFARE Classic * MIFARE Ultralight * NFC Forum NDEF tags   The platform also provides a limited peer-to-peer communication protocol and API. Foreground Activities can use the API to register an NDEF message that will get pushed to other NFC devices when they connect.  Advanced tag dispatching now gives applications more control over how and when they are launched, when an NFC tag is discovered. Previously, the platform used a single-step intent dispatch to notify interested applications that a tag was discovered. The platform now uses a four-step process that enables the foreground application to take control of a tag event before it is passed to any other applications (android.nfc.NfcAdapter.enableForegroundDispatch()). The new dispatch process also lets apps listen for specific tag content and tag technologies, based on two new intent actions — android.nfc.action.NDEF\_DISCOVERED and android.nfc.action.TECH\_DISCOVERED.  The NFC API is available in the android.nfc and android.nfc.tech packages. The key classes are:   * NfcAdapter, which represents the NFC hardware on the device. * NdefMessage, which represents an NDEF data message, the standard format in which "records" carrying data are transmitted between devices and tags. An NDEF message certain many NDEF records of different types. Applications can receive these messages from NDEF\_DISCOVERED, TECH\_DISCOVERED, or TAG\_DISCOVERED Intents. * NdefRecord, delivered in an NdefMessage, which describes the type of data being shared and carries the data itself. * Tag, which represents a tag scanned by the device. Multiple types of tags are supported, based on the underlying tag technology. * TagTechnology, an interface that gives applications access to tag properties and I/O operations based on the technologies present in the tag. For a full list of tag technologies supported in Android 2.3.3, see android.nfc.tech.   NFC communication relies on wireless technology in the device hardware, and is not present in all Android devices. Android devices that do not support NFC will return a null object when getDefaultAdapter(Context) is called, andcontext.getPackageManager().hasSystemFeature(PackageManager.FEATURE\_NFC) will return false. The NFC API is always present, however, regardless of underlying hardware support.  To use the NFC API, applications must request permission from the user by declaring <uses-permission android:name="android.permission.NFC"> in their manifest files.  Additionally, developers can request filtering on Google Play, such that their applications are not discoverable to users whose devices do not support NFC. To request filtering, add <uses-feature android:name="android.hardware.nfc" android:required="true"> to the application's manifest.  For more information, read the NFC developer guide. Bluetooth Android 2.3.3 adds platform and API support for Bluetooth nonsecure socket connections. This lets applications communicate with simple devices that may not offer a UI for authentication. SeecreateInsecureRfcommSocketToServiceRecord(java.util.UUID) andlistenUsingInsecureRfcommWithServiceRecord(java.lang.String, java.util.UUID) for more information. Graphics  * A new BitmapRegionDecoder class lets applications decode a rectangle region from an image. The API is particularly useful when an original image is large and and the application only need parts of the image. * A new inPreferQualityOverSpeed field in BitmapFactory.Options allows applications to use a more accurate but slightly slower IDCT method in JPEG decode. This in turn improves the quality of the reconstructed image.  Media framework  * A new MediaMetadataRetriever class provides a unified interface for retrieving frame and metadata from an input media file. * MediaRecorder.AudioEncoder and MediaRecorder.OutputFormat include new fields for specifying AMR Wideband and AAC formats.  Speech recognition The speech-recognition API includes new constants to let you manage voice search results in new ways. Although the new constants are not needed for normal use of speech recognition, you could use them to offer a different view of voice search results in your application. For information, see RecognizerResultsIntent. API Level The Android 2.3.3 platform delivers an updated version of the framework API. The Android 2.3.3 API is assigned an integer identifier — **10** — 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 2.3.3 in your application, you need compile the application against the Android library that is provided in the Android 2.3.3 SDK platform. Depending on your needs, you might also need to add anandroid:minSdkVersion="10" attribute to the <uses-sdk> element in the application's manifest. If your application is designed to run only on Android 2.3 and higher, declaring the attribute prevents the application from being installed on earlier versions of the platform. | | | |  | | |   **Dashboards**  **Google Play Install Stats**  The Google Play Developer Console also provides detailed statistics about your users' devices. Those stats may help you prioritize the device profiles for which you optimize your app.  This page provides information about the relative number of devices that share a certain characteristic, such as Android version or screen size. This information may help you prioritize efforts for supporting different devices by revealing which devices are active in the Android and Google Play ecosystem.  This data reflects devices running the latest Google Play Store app, which is compatible with Android 2.2 and higher. Each snapshot of data represents all the devices that visited the Google Play Store in the prior 7 days.  **Note:** Beginning in September, 2013, devices running versions older than Android 2.2 do not appear in this data because those devices do not support the new Google Play Store app. Only the new app is able to measure the number of devices that actively visit Google Play Store and we believe this measurement best reflects your potential user-base.  **Platform Versions**  This section provides data about the relative number of devices running a given version of the Android platform.  For information about how to target your application to devices based on platform version, read Supporting Different Platform Versions.   |  |  |  |  | | --- | --- | --- | --- | | Version | Codename | API | Distribution | | 2.2 | Froyo | 8 | 2.4% | | 2.3.3 - 2.3.7 | Gingerbread | 10 | 30.7% | | 3.2 | Honeycomb | 13 | 0.1% | | 4.0.3 - 4.0.4 | Ice Cream Sandwich | 15 | 21.7% | | 4.1.x | Jelly Bean | 16 | 36.6% | | 4.2.x | 17 | 8.5% |   http://chart.googleapis.com/chart?chs=500x250&cht=p&chco=c4df9b%2C6fad0c&chd=t%3A2.4%2C30.7%2C0.1%2C21.7%2C45.1&chf=bg%2Cs%2C00000000&chl=Froyo%7CGingerbread%7CHoneycomb%7CIce%20Cream%20Sandwich%7CJelly%20Bean  *Data collected during a 7-day period ending on September 4, 2013.  Any versions with less than 0.1% distribution are not shown.*  **Note:** Because this data is gathered from the new Google Play Store app, which supports Android 2.2 and above, devices running older versions are not included. However, in August, 2013, versions older than Android 2.2 accounted for about 1% of devices that *checked in* to Google servers (not those that actually visited Google Play Store).  **Screen Sizes and Densities**  This section provides data about the relative number of devices that have a particular screen configuration, defined by a combination of screen size and density. To simplify the way that you design your user interfaces for different screen configurations, Android divides the range of actual screen sizes and densities into several buckets as expressed by the table below.  For information about how you can support multiple screen configurations in your application, read Supporting Multiple Screens.   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | |  | ldpi | mdpi | tvdpi | hdpi | xhdpi | xxhdpi | **Total** | | Small | 9.5% |  |  |  |  |  | 9.5% | | Normal | 0.1% | 15.7% |  | 33.6% | 23.1% | 7.1% | 79.6% | | Large | 0.6% | 3.4% | 1.2% | 0.4% | 0.5% |  | 6.1% | | Xlarge |  | 4.4% |  | 0.3% | 0.1% |  | 4.8% | | **Total** | 10.2% | 23.5% | 1.2% | 34.3% | 23.7% | 7.1% |  |   http://chart.googleapis.com/chart?chs=400x250&cht=p&chco=c4df9b%2C6fad0c&chd=t%3A4.8%2C6.1%2C79.6%2C9.5&chf=bg%2Cs%2C00000000&chl=Xlarge%7CLarge%7CNormal%7CSmallhttp://chart.googleapis.com/chart?chs=400x250&cht=p&chco=c4df9b%2C6fad0c&chd=t%3A10.2%2C23.5%2C1.2%2C34.3%2C23.7%2C7.1&chf=bg%2Cs%2C00000000&chl=ldpi%7Cmdpi%7Ctvdpi%7Chdpi%7Cxhdpi%7Cxxhdpi  *Data collected during a 7-day period ending on September 4, 2013  Any screen configurations with less than 0.1% distribution are not shown.*  **Open GL Version**  This section provides data about the relative number of devices that support a particular version of OpenGL ES. Note that support for one particular version of OpenGL ES also implies support for any lower version (for example, support for version 2.0 also implies support for 1.1).  http://chart.googleapis.com/chart?cht=p&chs=400x250&chco=c4df9b,6fad0c&chl=GL%201.1%20only|GL%202.0%20%26%201.1&chd=t%3A0.2,99.8&chf=bg,s,00000000  To declare which version of OpenGL ES your application requires, you should use the android:glEsVersionattribute of the <uses-feature> element. You can also use the <supports-gl-texture> element to declare the GL compression formats that your application uses.   |  |  | | --- | --- | | OpenGL ES Version | Distribution | | 1.1 only | 0.2% | | 2.0 & 1.1 | 99.8% |   *Data collected during a 7-day period ending on September 4, 2013* | | |  | | | |
|  | |