

Android KitKat

Welcome to Android 4.4 KitKat!

Android KitKat brings all of Android's most innovative, most beautiful, and most useful features to more devices everywhere.

This document provides a glimpse of what's new for developers.

Find out more about KitKat for consumers at www.android.com



(<https://www.android.com/versions/kit-kat-4-4/>).

Making Android for everyone

Android 4.4 is designed to run fast, smooth, and responsively on a much broader range of devices than ever before — including on millions of entry-level devices around the world that have as little as **512MB RAM**.

KitKat streamlines every major component to reduce memory use and introduces new APIs and tools to help you create innovative, responsive, memory-efficient applications.

OEMs building the next generation of Android devices can take advantage of **targeted recommendations and options** to run Android 4.4 efficiently, even on low-memory devices. Dalvik JIT code cache tuning, kernel samepage merging (KSM), swap to zRAM, and other optimizations help manage memory. New configuration options let OEMs tune out-of-memory levels for processes, set graphics cache sizes, control memory reclaim, and more.

memory levels for processes, set graphics cache sizes, control memory reclaim, and more.

In Android itself, changes across the system improve memory management and reduce memory footprint. Core system processes are trimmed to **use less heap**, and they now more **aggressively protect system memory** from apps consuming large amounts of RAM. When multiple services start at once — such as when network connectivity changes — Android now **launches the services serially**, in small groups, to avoid peak memory demands.

For developers, Android 4.4 helps you deliver **apps that are efficient and responsive** on all devices. A new API, `ActivityManager.isLowRamDevice()`, lets you tune your app's behavior to match the device's memory configuration. You can modify or disable large-memory features as needed, depending on the use-cases you want to support on entry-level devices. Learn more about optimizing your apps for low-memory devices [here](/training/articles/memory) (/training/articles/memory).

New tools also give you powerful insight into your app's memory use. The **procstats tool** details memory use over time, with run times and memory footprint for foreground apps and background services. An on-device view is also available as a new developer option. The **meminfo tool** is enhanced to make it easier to spot memory trends and issues, and it reveals additional memory overhead that hasn't previously been visible.

New NFC capabilities through Host Card Emulation

Android 4.4 introduces new platform support for secure NFC-based transactions through **Host Card Emulation** (HCE), for payments, loyalty programs, card access, transit passes, and other custom services. With HCE, any app on an Android device can emulate an NFC smart card, letting users tap to initiate transactions with an app of their choice — no provisioned secure element (SE) in the device is needed. Apps can also use a new **Reader Mode** to act as readers for HCE cards and other NFC-based transactions.

Android HCE emulates ISO/IEC 7816 based smart cards that use the contactless ISO/IEC 14443-4 (ISO-DEP) protocol for transmission. These cards are used by many systems today, including the existing EMVCO NFC payment infrastructure. Android uses Application Identifiers (AIDs) as defined in ISO/IEC 7816-4 as the basis for routing transactions to the correct Android applications.



Apps declare the AIDs they support in their manifest files, along with a category identifier that indicates the type of support available (for example, "payments"). In cases where multiple apps support the same AID in the same category, Android displays a dialog that lets the user choose which app to use.

When the user taps to pay at a point-of-sale terminal, the system extracts the preferred AID and routes the transaction to the correct application. The app reads the transaction data and can use any local or network-based services to verify and then complete the transaction.

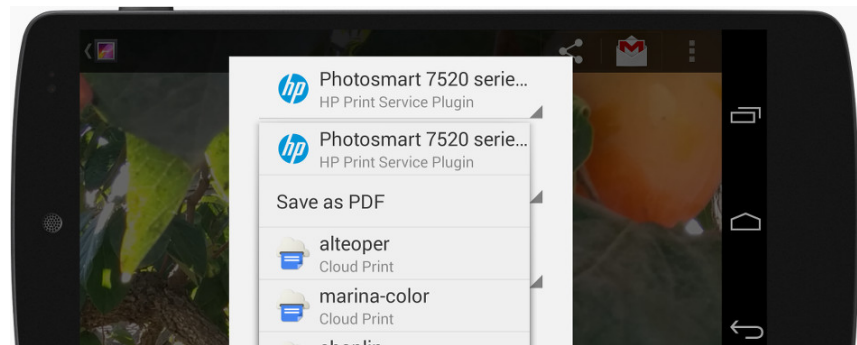
Android HCE requires an NFC controller to be present in the device. Support for HCE is already widely available on most NFC controllers, which offer dynamic support for both HCE and SE transactions. Android 4.4 devices that support NFC will include Tap & Pay for easy payments using HCE.

Printing framework

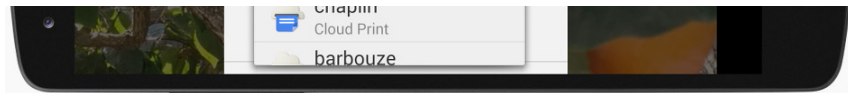
Android apps can now print any type of content over Wi-Fi or cloud-hosted services such as Google Cloud Print. In print-enabled apps, users can discover available printers, change paper sizes, choose specific pages to print, and print almost any kind of document, image, or file.

Android 4.4 introduces native platform support for printing, along with APIs for managing printing and adding new types of printer support. The platform provides a print manager that mediates between apps requesting printing and installed print services that handle print requests. The print manager provides shared services and a system UI for printing, giving users consistent control over printing from any app. The print manager also ensures the security of content as it's passed across processes, from an app to a print service.

Printer manufacturers can use new APIs to develop their own **print services** — pluggable components that add vendor-specific logic and services for



communicating with specific types of printers. They can build print services and distribute them



You can add printing support to your apps or develop print services to support specific types of printers.

through Google Play, making it easy for users to find and install them on their devices. Just as with other apps, you can update print services over-the-air at any time.

Client apps can use new APIs to add printing capabilities to their apps with minimal code changes. In most cases, you would add a print action to your Action Bar and a UI for choosing items to print. You would also implement APIs to create print jobs, query the print manager for status, and cancel jobs. This lets you print nearly any type of content, from local images and documents to network data or a view rendered to a canvas.

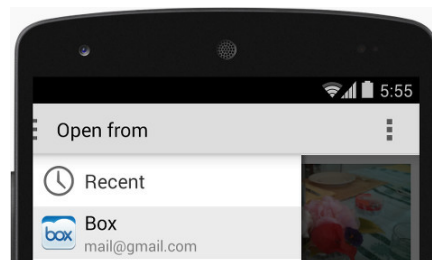
For broadest compatibility, Android uses PDF as its primary file format for printing. Before printing, your app needs to generate a properly paginated PDF version of your content. For convenience, the printing API provides native and WebView helper classes to let you create PDFs using standard Android drawing APIs. If your app knows how to draw the content, it can quickly create a PDF for printing.

Most devices running Android 4.4 will include Google Cloud Print pre-installed as a print service, as well as several Google apps that support printing, including Chrome, Drive, Gallery, and QuickOffice.

Storage access framework

A new **storage access framework** makes it simple for users to browse and open documents, images, and other files across all of their preferred document storage providers. A standard, easy-to-use UI lets users browse files and access recents in a consistent way across apps and providers.

Cloud or local storage services can participate in this ecosystem by implementing a new document provider class that



encapsulates their services. The provider class includes all of the APIs needed to register the provider with the system and manage browsing, reading, and writing documents in the provider. The document provider can give users access to any remote or local data that can be represented as files — from text, photos, and wallpapers to video, audio, and more.

If you build a **document provider** for a cloud or local service, you can deliver it to users as part of your existing Android app. After downloading and installing the app, users will have instant access to your service from any app that participates in the framework. This can help you gain exposure and user engagement, since users will find your services more easily.



Box and others have integrated their services into the storage access framework, giving users easy access to their documents from apps across the system.

If you develop a **client app** that manages files or documents, you can integrate with the storage access framework just by using new `CREATE_DOCUMENT` or `OPEN_DOCUMENT` intents to open or create files — the system automatically displays the standard UI for browsing documents, including all available document providers.

You can integrate your client app one time, for all providers, without any vendor-specific code. As users add or remove providers, they'll continue to have access to their preferred services from your app, without changes or updates needed in your code.

The storage access framework is integrated with the existing `GET_CONTENT` intent, so users also have access to all of their previous content and data sources from the new system UI for browsing. Apps can continue using `GET_CONTENT` as a way to let users import data. The storage access framework and system UI for browsing make it easier for users to find and import their data from a wider range of sources.

Most devices running Android 4.4 will include Google Drive and local storage pre-integrated as document providers, and Google apps that work with files also use the new framework.

Low-power sensors

Sensor batching

Android 4.4 introduces platform support for **hardware sensor batching**, a new optimization that can dramatically reduce power consumed by ongoing sensor activities.

With sensor batching, Android works with the device hardware to collect and deliver sensor events efficiently in batches, rather than individually as they are detected. This lets the device's application processor remain in a low-power idle state until batches are delivered. You can request batched events from any sensor using a standard event listener, and you can control the interval at which you receive batches. You can also request immediate delivery of events between batch cycles.

Sensor batching is ideal for low-power, long-running use-cases such as fitness, location tracking, monitoring, and more. It can make your app more efficient and it lets you track sensor events continuously — even while the screen is off and the system is asleep.

Sensor batching is currently available on Nexus 5, and we're working with our chipset partners to bring it to more devices as soon as possible.

Step Detector and Step Counter

Android 4.4 also adds platform support for two new composite sensors — step detector and step counter — that let your app track steps when the user is walking, running, or climbing stairs. These new sensors are implemented in hardware for low power consumption.

The step detector analyzes accelerometer input to recognize when the user has taken a step, then triggers an event with each step. The step counter tracks the total number of steps since the last device reboot and triggers an event



with each change
in the step count.
Because the logic
and sensor
management is



Moves and **Runtastic Pedometer** are using the hardware step-detector to offer long-running, low-power services.

built into the platform and underlying hardware, you don't need to maintain your own detection algorithms in your app.

Step detector and counter sensors are available on Nexus 5, and we're working with our chipset partners to bring them to new devices as soon as possible.

SMS provider

If you develop a messaging app that uses SMS or MMS, you can now use a **shared SMS provider and new APIs** to manage your app's message storage and retrieval. The new SMS provider and APIs define a standardized interaction model for all apps that handle SMS or MMS messages.

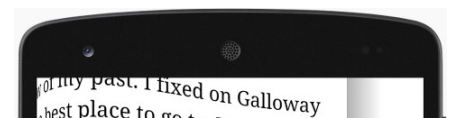
Along with the new provider and APIs, Android 4.4 introduces **new semantics** for receiving messages and writing to the provider. When a message is received, the system routes it directly to the user's default messaging app using the new `SMS_DELIVER` intent. Other apps can still listen for incoming messages using the `SMS_RECEIVED` intent. Also, the system now allows only the default app to write message data to the provider, although other apps can read at any time. Apps that are not the user's default can still send messages — the system handles writing those messages to the provider on behalf of the app, so that users can see them in the default app.

The new provider and semantics help to improve the user's experience when multiple messaging apps are installed, and they help you to build new messaging features with fully-supported, forward-compatible APIs.

New ways to build beautiful apps

Full-screen Immersive mode

Now your apps can use **every pixel on the device**



Now your apps can use every pixel on the device **screen** to showcase your content and capture touch events. Android 4.4 adds a new full-screen immersive mode that lets you create full-bleed UIs reaching from edge to edge on phones and tablets, **hiding all system UI** such as the status bar and navigation bar. It's ideal for rich visual content such as photos, videos, maps, books, and games.

In the new mode, the system UI stays hidden, even while users are interacting with your app or game — you can capture touch events from anywhere across the screen, even areas that would otherwise be occupied by the system bars. This gives you a great way to create a larger, richer, more immersive UI in your app or game and also reduce visual distraction.

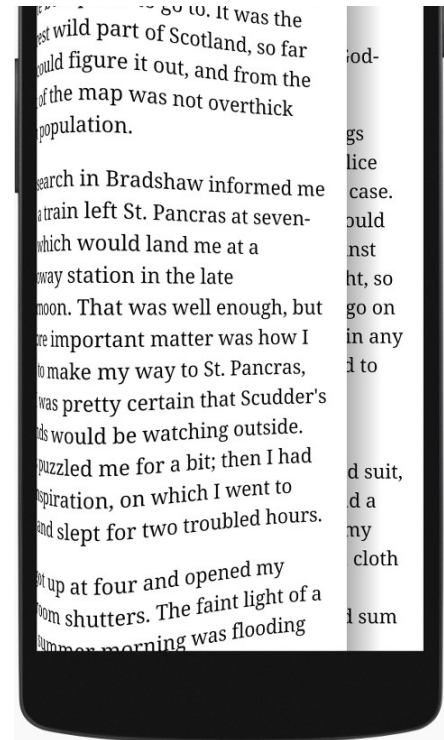
To make sure that users always have easy, consistent access to system UI from full-screen immersive mode, Android 4.4 supports a new gesture — in immersive mode, an edge swipe from the top or bottom of the screen now reveals the system UI.

To return to immersive mode, users can touch the screen outside of the bar bounds or wait for a short period for the bars to auto-hide. For a consistent user experience, the new gesture also works with previous methods of hiding the status bar.

Transitions framework for animating scenes

Most apps structure their flows around several key UI states that expose different actions. Many apps also use animation to help users understand their progress through those states and the actions available in each. To make it easier to create **high-quality animations** in your app, Android 4.4 introduces a new transitions framework.

The transitions framework lets you define **scenes**, typically view hierarchies, and transitions, which describe how to animate or transform the scenes when the user enters or exits them. You can use several predefined transition types to animate your scenes based on specific properties, such as layout bounds, or visibility. There's also an auto-transition type that automatically fades, moves, and resizes views during a scene change. In addition, you can define custom transitions that animate the properties that matter most to your app, and you



A new **immersive mode** lets apps use every pixel on the screen to show content and capture touch events.

can plug in your own animation styles if needed.

With the transitions framework you can also **animate changes to your UI on the fly**, without needing to define scenes. For example, you can make a series of changes to a view hierarchy and then have the TransitionManager automatically run a delayed transition on those changes.

Once you've set up transitions, it's straightforward to invoke them from your app. For example, you can call a single method to begin a transition, make various changes in your view hierarchy, and on the next frame animations will automatically begin that animate the changes you specified.

For custom control over the transitions that run between specific scenes in your application flow, you can use the TransitionManager. The TransitionManager lets you define the relationship between scenes and the transitions that run for specific scene changes.



Apps can use new window styles to request translucent system bars.

Translucent system UI styling

To get the most impact out of your content, you can now use new window styles and themes to request **translucent system UI**, including both the status bar and navigation bar. To ensure the legibility of navigation bar buttons or status bar information, subtle gradients is shown behind the system bars. A typical use-case would be an app that needs to show through to a wallpaper.

Enhanced notification access

Notification listener services can now see **more information about incoming notifications** that were constructed using the notification builder APIs. Listener services can access a notification's actions as well as new extras fields — text, icon, picture, progress, chronometer, and many others — to extract cleaner information about the notification and present the information in a different way.



()

Chromium WebView

Android 4.4 includes a completely new implementation of WebView that's based on Chromium (<http://www.chromium.org/Home>). The new Chromium WebView gives you the latest in standards support, performance, and compatibility to build and display your web-based content.

Chromium WebView provides broad support for HTML5, CSS3, and JavaScript. It supports most of the HTML5 features available in Chrome for

Android 30. It also brings an updated version of the JavaScript Engine (V8) that delivers dramatically improved JavaScript performance.

In addition, the new Chromium WebView supports remote debugging using Chrome DevTools

(<https://developers.google.com/chrome-developer-tools/docs/remote-debugging#debugging-webviews>)

. For example, you can use Chrome DevTools on your development machine to inspect, debug, and analyze your WebView content live on a mobile device.

The new Chromium WebView is included on all compatible devices running Android 4.4 and higher. You can take advantage of the new WebView right away, and with minimum modifications to existing apps and content. In most cases, your content will migrate to the new implementation seamlessly.

New media capabilities

Screen recording

Now it's easy to create high-quality video of your app, directly from your Android device. Android 4.4 adds support for screen recording and provides a **screen recording utility** that lets you start and stop recording on a device that's connected to your Android SDK environment over USB. It's a great new way to create walkthroughs and tutorials for your app, testing materials, marketing videos, and more.

With the screen recording utility, you can capture video of your device screen contents and

store the video as an MP4 file on the device. You can record at any device-supported resolution and bitrate you want, and the output retains the aspect ratio of the display. By default, the utility selects a resolution equal or close to the device's display resolution in the current orientation. When you are done recording, you can share the video directly from your device or pull the MP4 file to your host computer for post-production.

If your app plays video or other protected content that you don't want to be captured by the screen recorder, you can use `SurfaceView.setSecure()` to mark the content as secure.

You can access screen recording through the adb tool included in the Android SDK, using the command `adb shell screenrecord`. You can also launch it through logcat (`/studio/debug/am-logcat`) in Android Studio.

Resolution switching through adaptive playback

Android 4.4 brings formal support for adaptive playback into the Android media framework. Adaptive playback is an optional feature of video decoders for MPEG-DASH and other formats that enables **seamless change in resolution during playback**. The client can start to feed the decoder input video frames of a new resolution and the resolution of the output buffers change automatically, and without a significant gap.

Resolution switching in Android 4.4 lets media apps offer a significantly better streaming video experience. Apps can check for adaptive playback support at runtime using existing APIs and implement resolution-switching using new APIs introduced in Android 4.4.

Common Encryption for DASH

Android now supports the **Common Encryption (CENC)** for MPEG-DASH, providing a standard, multiplatform DRM scheme for managing protecting content. Apps can take advantage of CENC through Android's modular DRM framework and platform APIs for supporting DASH.

HTTP Live Streaming

Android 4.4 updates the platform's HTTP Live Streaming (HLS) support to a superset of version 7 of the HLS specification (version 4 of the protocol). See the IETF draft (<http://tools.ietf.org/html/draft-pantos-http-live-streaming-07>) for details.

Audio Tunneling to DSP

For high-performance, lower-power audio playback, Android 4.4 adds platform support for audio tunneling to a digital signal processor (DSP) in the device chipset. With tunneling, audio decoding and output effects are off-loaded to the DSP, waking the application processor less often and using less battery.

Audio tunneling can **dramatically improve battery life** for use-cases such as listening to music over a headset with the screen off. For example, with audio tunneling, Nexus 5 offers a total off-network audio playback time of up to 60 hours, an increase of over 50% over non-tunneled audio.

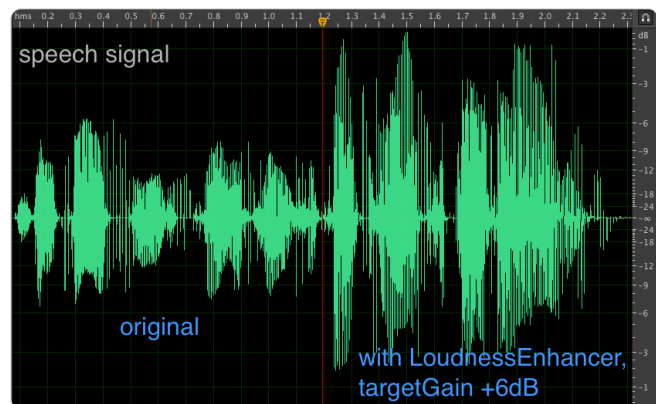
Media applications can take advantage of audio tunneling on supported devices without needing to modify code. The system applies tunneling to optimize audio playback whenever it's available on the device.

Audio tunneling requires support in the device hardware. Currently audio tunneling is available on Nexus 5 and we're working with our chipset partners to make it available on more devices as soon as possible.

Audio monitoring

Apps can use new monitoring tools in the Visualizer effect to get updates on the **peak and RMS**

levels of any currently playing audio on the device. For example, you could use this creatively in music visualizers or to implement playback metering in a media player.



Visualization of how the LoudnessEnhancer effect can make speech content more audible.

Loudness enhancer

Media playback applications can **increase the loudness of spoken content** by using the new LoudnessEnhancer effect, which acts as compressor with time constants that are specifically tuned for speech.

Audio timestamps for improved AV sync

The audio framework can now report **presentation timestamps** from the audio output HAL to applications, for better audio-video synchronization. Audio timestamps let your app determine when a specific audio frame will be (or was) presented off-device to the user; you can use the timestamp information to more accurately synchronize audio with video frames.

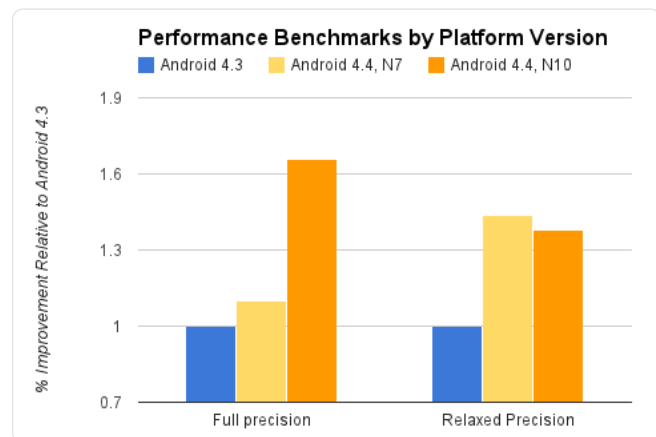
Wi-Fi CERTIFIED Miracast™

Android 4.4 devices can now be certified to the Wi-Fi Alliance Wi-Fi Display Specification as Miracast compatible. To help with testing, a new Wireless Display developer option exposes advanced configuration controls and settings for Wireless Display certification. You can access the option at **Settings > Developer options > Wireless display certification**. Nexus 5 is a Miracast certified wireless display device.

RenderScript Compute

Ongoing performance improvements

When your apps use RenderScript, they'll benefit from **ongoing performance tuning** in the RenderScript runtime itself, without the need for recompilation. The chart at right shows performance gains in Android 4.4 on two popular chipsets.



Performance benchmarks for Android 4.4 relative to Android 4.3, run on the same devices (Nexus 7, Nexus 10).

GPU acceleration

Any app using RenderScript on a supported device benefits from GPU acceleration, without code changes or recompiling. Since the Nexus 10 first debuted RenderScript GPU acceleration, various other hardware partners have added support.

Now with Android 4.4, GPU acceleration is available on the Nexus 5, as well as the Nexus 4, Nexus 7 (2013), and Nexus 10, and we're working with our partners to bring it to more devices as soon as possible.

devices as soon as possible.

RenderScript in the Android NDK

Now you can take advantage of RenderScript **directly from your native code**. A new C++ API in the Android Native Development Kit (NDK) lets you access the same RenderScript functionality available through the framework APIs, including script intrinsics, custom kernels, and more.

If you have large, performance-intensive tasks to handle in native code, you can perform those tasks using RenderScript and integrate them with your native code. RenderScript offers great performance across a wide range of devices, with automatic support for multi-core CPUs, GPUs, and other processors.

When you build an app that uses the RenderScript through the NDK, you can distribute it to any device running Android 2.2 or higher, just like with the RenderScript support library available for framework APIs.

Graphics

GL ES 2.0 SurfaceFlinger

Android 4.4 upgrades its SurfaceFlinger from OpenGL ES 1.0 to OpenGL ES 2.0.

New Hardware Composer support for virtual displays

The latest version of Android Hardware Composer, HWComposer 1.3, supports hardware composition of one virtual display in addition to the primary, external (e.g. HDMI) display, and has improved OpenGL ES interoperability.

New Types of Connectivity

New Bluetooth profiles

Android 4.4 support for two new Bluetooth profiles to let apps support a broader range of low-power and media interactions. **Bluetooth HID over GATT** (HOGP) gives apps a low-latency link with low-power peripheral devices such as mice, joysticks, and keyboards.

latency link with low-power peripheral devices such as mice, joysticks, and keyboards.

Bluetooth MAP lets your apps exchange messages with a nearby device, for example an automotive terminal for handsfree use or another mobile device. As an **extension to Bluetooth AVRCP 1.3**, users can now set absolute volume on the system from their Bluetooth devices.

Platform support for HOGP, MAP, and AVRCP is built on the 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 compatible Bluetooth capabilities.

IR Blasters

Android 4.4 introduces platform support for built-in **IR blasters**, along with a new API and system service that let you create apps to take advantage them.

Using the new API, you can build apps that let users remotely control nearby TVs, tuners, switches, and other electronic devices. The API lets your app check whether the phone or tablet has an infrared emitter, query its carrier frequencies, and then send infrared signals.

Because the API is standard across Android devices running Android 4.4 or higher, your app can support the broadest possible range of vendors without writing custom integration code.

Wi-Fi TDLS support

Android 4.4 introduces a seamless way to stream media and other data faster between devices already on the same Wi-Fi network by supporting Wi-Fi Tunneled Direct Link Setup (TDLS).

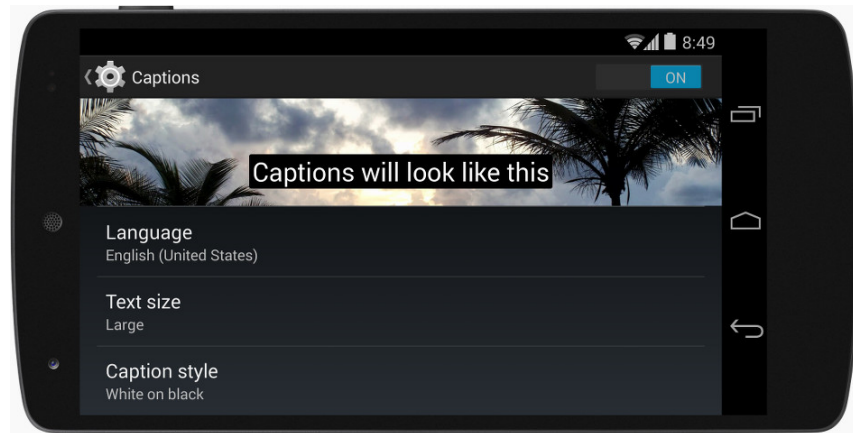
Accessibility

System-wide settings for closed captioning

Android 4.4 now supports a better accessibility experience across apps by adding system-wide preferences for Closed Captioning. Users can go to **Settings > Accessibility > Captions** to set global captioning preferences, such as whether to show captions and what language, text size, and text style to use.

Apps that use video can now access the user's captioning settings and **adjust presentation to meet the user's preferences**. A new captioning manager API lets you check and monitor the user's captioning preferences. The captioning manager provides you with the user's preferred captioning state as well as preferred locale, scaling factor, and text style. The text style includes foreground and background colors, edge properties, and typeface.

In addition, apps that use **VideoView** can use a new API to pass a captioning stream along with a video stream for rendering. The system automatically handles the display of the captions on video frames according to the user's systemwide settings. Currently, VideoView supports auto-display of captions in WebVTT format only.



Apps can now refer to the user's **system-wide captions preferences**. An example of the expected display style is shown right in the settings.

All apps that show captions should make sure to check the user's systemwide captioning preferences and render captions as closely as possible to those preferences. For more insight into how specific combinations of settings should look, you can look at a preview of captions in different languages, sizes, and styles right in the Settings app.

Enhanced Accessibility APIs

Android 4.4 extends the accessibility APIs to support **more precise structural and semantic description** and observation of onscreen elements. With the new APIs, developers can improve the quality of accessible feedback by providing accessibility services with more information about on-screen elements.

In accessibility nodes, developers can now determine whether a node is a popup, get its input type, and more. You can also use new APIs to work with nodes that contain grid-like information, such as lists and tables. For example, you can now specify new supported actions, collection information, live region modes, and more.

New accessibility events let developers more closely follow the changes that are taking place in window content, and they can now listen for changes in the touch exploration mode on the device.

Support for international Users

Drawable mirroring for RTL locales

If your app is targeting users who use RTL scripts, you can use a new API to declare that a **drawable should be auto-mirrored** when the user's locale setting includes an RTL language.

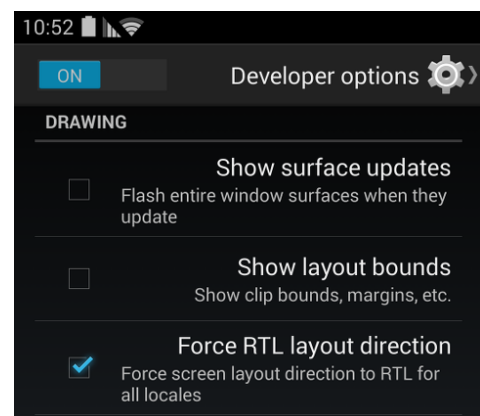
Declaring a drawable as auto-mirrored helps you **prevent duplication of assets** in your app and reduces the size of your APK. When you have drawables that are the reusable for both LTR and RTL presentations, you can declare the default versions as auto-mirrored and then omit those Drawables from your RTL resources.

You can declare various types of drawables as auto-mirrored in your application code, such as bitmap, nine-patch, layer, state list, and other drawables. You can also declare a drawable as auto-mirrored in your resource files by using a new attribute.

Force RTL Layout

To make it easier to test and debug layout mirroring issues without switching to an RTL language, Android includes a new developer option to force RTL layout direction in all apps.

The Force RTL layout option switches the device to RTL layout for all locales and displays text in your current language. This can help you find layout issues across your app, without having to display the app in an RTL language. You can access the option in **Settings > Developer options > Force RTL layout direction**.



The **Force RTL layout** option makes it easier to test your app's localization.

Security enhancements

SELinux (enforcing mode)

Android 4.4 updates its SELinux configuration from "permissive" to "enforcing." This means potential policy violations within a SELinux domain that has an enforcing policy will be blocked.

Improved cryptographic algorithms

Android has improved its security further by adding support for two more cryptographic algorithms. Elliptic Curve Digital Signature Algorithm (ECDSA) support has been added to the keystore provider improving security of digital signing, applicable to scenarios such as signing of an application or a data connection. The Scrypt key derivation function is implemented to protect the cryptographic keys used for full-disk encryption.

Other enhancements

On multiuser devices, VPNs are now applied per user. This can allow a user to route all network traffic through a VPN without affecting other users on the device. Also, Android now supports FORTIFY_SOURCE level 2, and all code is compiled with those protections. FORTIFY_SOURCE has been enhanced to work with clang.

Tools for analyzing memory use

Procstats

A new tool called **procstats** helps you analyze the memory resources your app uses, as well as the resources used by other apps and services running on the system.

Procstats keeps track of **how apps are running over time**, providing data about their execution durations and memory use to help determine how efficiently they are performing. This is most important for apps that start services that run in the background, since it lets you monitor how long they are running and how much RAM they are using while doing so. Procstats will also collect data for foreground applications about memory use over time to determine the overall memory profile of the app.

Procstats can help you identify background services started by your app. You can keep track of how long those services continue running and how much RAM they use while doing so. Procstats also lets you profile your app while it's in the foreground, using its memory use over time to determine its overall memory profile.

```

Home: 0.00%
(Cached): 67% (54MB-56MB-67MB/44MB-47MB-62MB over 32)
* com.google.android.gms / u0a7:
  TOTAL: 28% (10MB-11MB-11MB/8.0MB-8.1MB-8.2MB over 16)
  Top: 27% (10MB-11MB-11MB/8.0MB-8.1MB-8.2MB over 16)
  Imp Fg: 1.2%
  Service: 0.14%
  Receiver: 0.01%
  Home: 0.00%
  (Last Act): 0.09%
  (Cached): 70% (10MB-11MB-11MB/8.0MB-8.1MB-8.2MB over 15)
* com.google.android.apps.books / u0a26:
  TOTAL: 22% (50MB-80MB-113MB/43MB-63MB-82MB over 12)
  Top: 22% (50MB-80MB-113MB/43MB-63MB-82MB over 12)
  Service: 0.00%
  (Last Act): 4.6% (85MB-85MB-85MB/77MB-77MB-77MB over 2)
  (Cached): 58% (78MB-81MB-82MB/70MB-73MB-74MB over 4)
* com.android.settings / 1000:
  TOTAL: 2.8% (14MB-15MB-16MB/11MB-12MB-13MB over 3)
  Top: 2.8% (14MB-15MB-16MB/11MB-12MB-13MB over 3)

```

The new **procstats** tool lets you check the memory use of apps and services over time.

```

261 kB: bridgemgrd (pid 183)
236 kB: dhcpcd (pid 13422)
232 kB: adbd (pid 4572)
227 kB: debuggerd (pid 172)
199 kB: rmt_storage (pid 170)
194 kB: qseecomd (pid 252)
188 kB: mpdecision (pid 1466)
183 kB: installd (pid 178)
140 kB: healthd (pid 167)
114 kB: servicemanager (pid 168)
103 kB: qseecomd (pid 193)
94 kB: logwrapper (pid 990)

Total PSS by OOM adjustment:
43529 kB: Native
12363 kB: mediaserver (pid 177)
5889 kB: mm-qcamera-daemon (pid 195)
5480 kB: sensors.qcom (pid 187)
3842 kB: surfaceflinger (pid 174)
3488 kB: zygote (pid 175)

```

The enhanced **meminfo** tool lets you see device memory use for an app.

You can access procstats from the adb tool included in the Android SDK, `adb shell dumpsys procstats`. Also, for on-device profiling, see the Process Stats developer option, below.

On-device memory status and profiling

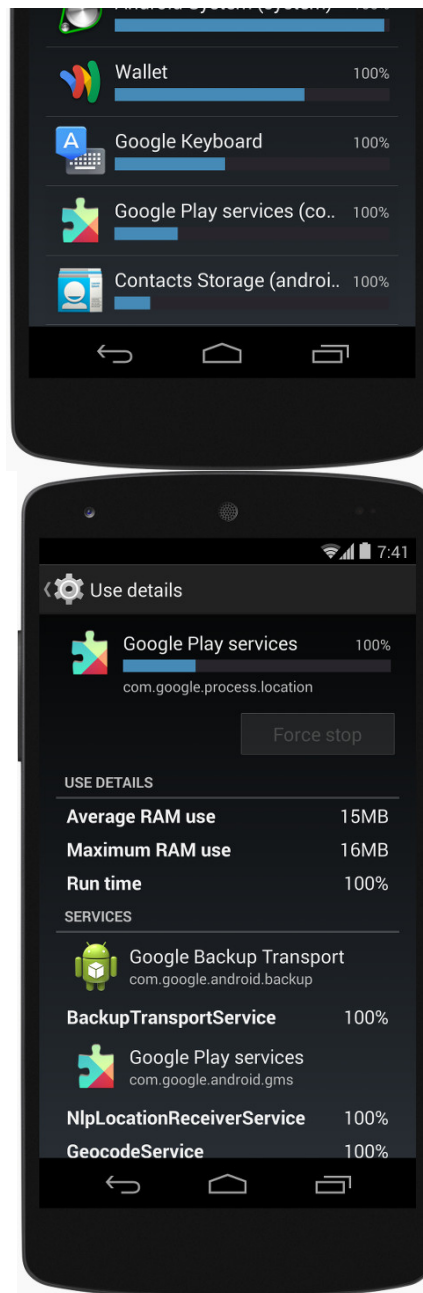
Android 4.4 includes a new developer option to make it easier to analyze your app's memory profile while it's running on any device or emulator. It's especially useful to get a view of how your app uses memory and performs on devices with low RAM. You can access the option at **Settings > Developer options > Process stats**

The **Process Stats** option shows you a variety of high-level metrics on your app's memory use, based on data collected using the new procstats service. On the main screen you can see a summary



of system memory status. Green indicates relative amount of time spent with low RAM usage, yellow indicates moderate RAM usage, and red indicates high (critical) RAM usage

Below the summary is a list summarizing each app's **memory load on the system**. For each app, a blue bar indicates the relative computed memory load (runtime x avg_pss) of its process, and a percentage number indicates the relative amount of time spent in the background. You can filter the list to show only foreground, background, or cached processes, and you can include or exclude system processes. You can also change the duration of the data collected to 3, 6, 12, or 24 hours, and you can include or exclude user memory.



Process stats is a convenient way to check your app's memory use. You can see how your app compares to other apps and zoom in on specific data about your app or its background services.

To take a closer look at a specific app's memory usage in isolation, tap the app. For each app, you can now see a summary of the memory consumed and the percentage of the

app, you can now see a summary of the memory consumed and the percentage of the collection interval that the app has been running. You can also see the average and maximum usage over the collection period, and below the app's services and the percentage of time they've been running.

Analyzing your app using the data in Process Stats can reveal issues and suggest possible optimizations for your app. For example, if your app is running longer than it should or using too much memory over a period of time, there could be bugs in your code that you can resolve to improve your app's performance, especially when running on a device with low RAM.

Content and code samples on this page are subject to the licenses described in the [Content License \(/license\)](#). Java and OpenJDK are trademarks or registered trademarks of Oracle and/or its affiliates.

Last updated 2021-03-11 UTC.