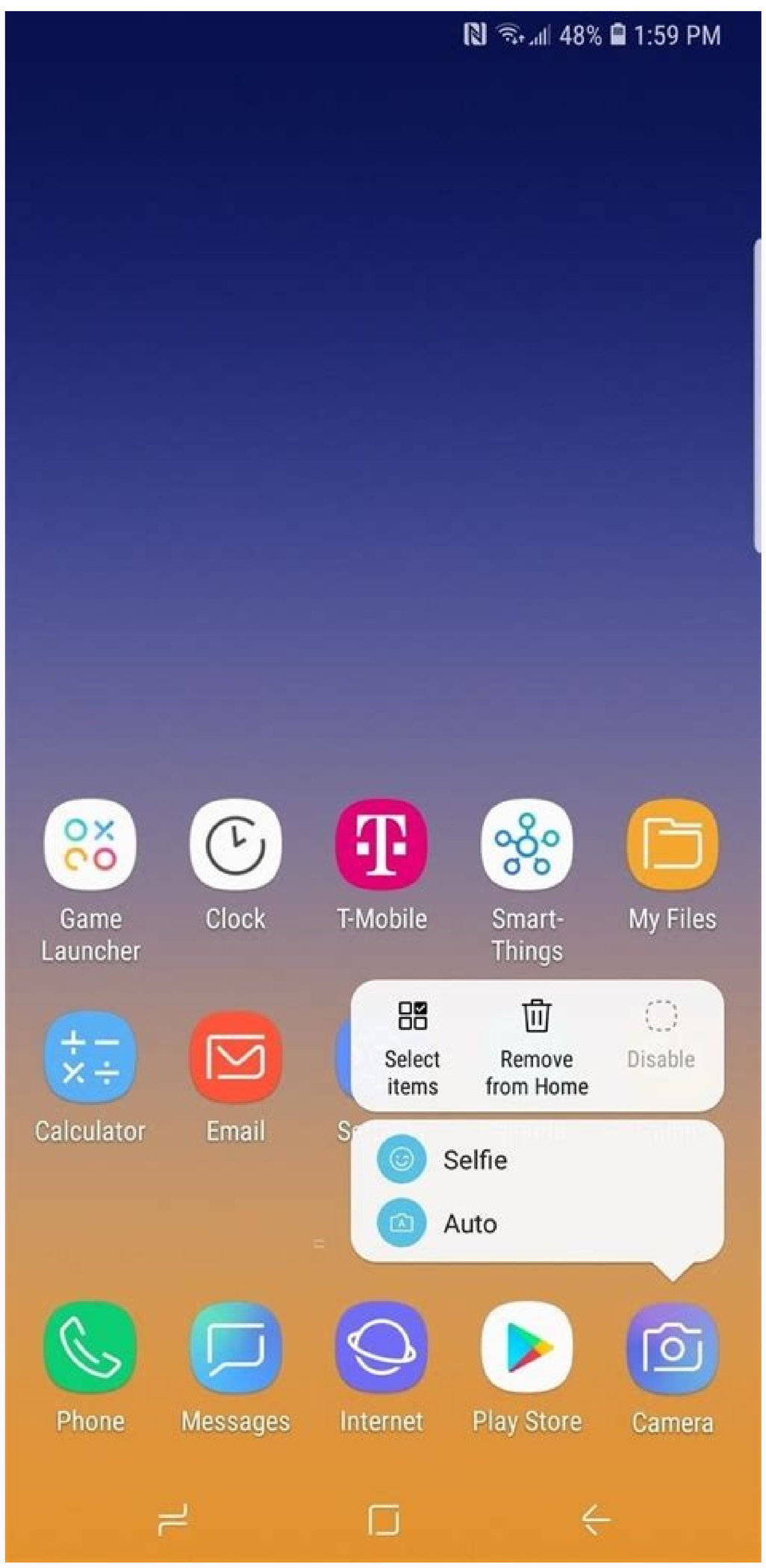
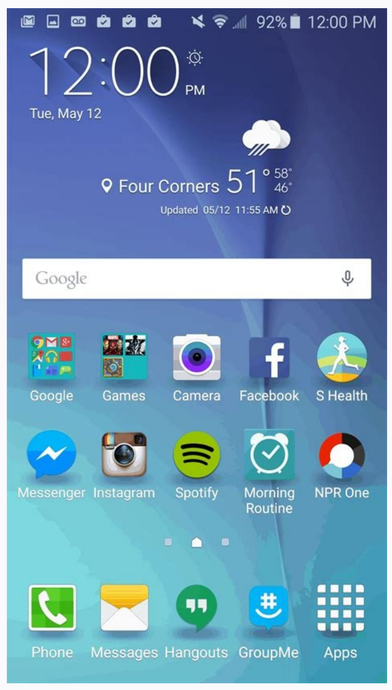
Multiple videos in one screen android

Continue


Grid
NEXT
Add Multiple Videos in
One Screen Easily


LTE
8:24
kindle
Amazon Kindle
Calculator
31
Calendar
Camera
Chrome
Clock
DI
DI Radio
Downloads
Drive
Earth
Email
feedly
Gallery
Gmail
Google
Google Settings
Google+
Hangouts
Instagram
Keep





Many users have been experiencing troubles in combining multiple videos into one screen similar to Zoom. Let’s say, you are making a video for your band of four people, and each one of you records a portion of the song, and you want to make a 4-way split-screen video. Commonly you need an experienced video editor using sophisticated video editors to complete the task. But what if you are on a budget? In this article, we will explore the top 6 ways to combine multiple videos into one screen. Part 1. Overall Best Software to Create Multi-Screen Videos Tipard Video Converter Ultimate If you are searching for a video maker software for Mac/PC that could put multiple videos together, you either have to learn complicated skills with highly sophisticated software or unusable applications. Tipard Video Converter Ultimate stands right on the sweet spot balancing the two poles. You will take almost no time from picking up the basics to produce a well-made multi-screen video. Key Features – User-friendly and sleek design. – Creative pre-made video templates. – Powerful built-in video editing tools. – Export to practically all popular video formats. – HD/8K/4K video resolution support. Overall Ratings: ★★★★★ How to Put Multiple Videos Together on Mac/PC Step 1Download Tipard Video Converter Ultimate. After installation, open the program and click on the Collage tab on the ribbon. Step 2On the next screen, select any desirable split-screen effect template. Step 3Keep in mind that the boundaries of each mini-video are adjustable. Drag the boundary line of any video to change the structure of your video collage. Step 4If you wish to add in more after effects, click on the Filters tab to give the video a different look. A quick preview is available on the right side of the screen. Step 5Move to the Export tab, set relevant variables, and click on the Export button to save the final output. Part 2. Other Apps to Combine Videos on iPhone/Android iMovie for iOS One of the top-rated video editing apps on the App Store brought to you by Apple. Don’t expect a more streamlined-designed app that fits your need to combine multiple videos into one screen. Key Features – Exceptional video editing tools. – Effortlessly transfer videos between Apple devices and onto iCloud. – Frequent updates and strong support from developers. – Split screen to create multiple screens. Pricing Free Bottom Line You could hardly find any app that’s better than iMovie on iPhone, considering it’s free. However, it falls short for supporting iOS devices only. Overall Ratings: ★★★★☆ How to Combine Video on iPhone with iMovie Step 1Download iMovie App from the Apple App Store. Open the iMovie app. Step 2On the main screen of the app, tap on the Create Project option. You would be asked the type of project that you intend to create. Choose the Movie option instead of the Trailer option. Step 3Import the videos from your Photos Library. Tap on the Create Movie option at the bottom of the screen to proceed. Step 4Now your multiple movies would be inserted into the iMovie timeline. Edit the movies with built-in effects to put multiple videos into one screen. Step 5Once you have finished editing the video, tap on the Share button at the bottom of the screen. Tap on the Save Video option on the menu to save your video to the Photos Library. Adobe Premiere Rush Standing out among the iPhone editing apps for its name recognition, Adobe Premiere Rush is more watered down and touchscreen-friendly Adobe Premiere software that could put multiple videos into one screen. Key Features – Easily Import, rearrange, merge multiple video clips. – Tablet-optimized user interface. – Multiple motion graphics templates are available. – Preview Videos before exporting. – Upload to Adobe Creative Cloud. – Supports iOS and Android. Pricing For $9.99 per month, you get unlimited access to 100GB cloud storage, which also gains you access to Adobe Creative Cloud Libraries and Assets with premade motion graphic assets and patterns. Bottom Line As a beginner’s video editor, Adobe Premiere Rush does its job well for editing across mobile and desktop devices. However, you might consider charging a $9.9 per month fee doesn’t merit for one single mobile app. Overall Ratings: ★★★★☆ Part 3. FAQs about How to Put Multiple Videos Together 1. Is there a way to combine 4 videos into one screen on Android? Alight Motion is an equivalent mobile app to professional Adobe software. However, keep in mind it leaves a watermark on your final output. 2. Is there an open-source video editing app to put multiple videos into one screen? Popular open-source video solution software FFmpeg does have such capacity. It has a filtering library that could create overlays and allow users to put one video over another. As such, you could create a multiple-screen video with 4 or even more components altogether. However, it requires a good knowledge of using the command line, thus it may be not suitable for beginner users. 3. Which Multiple Video Collage App is the best? It depends on your particular use case. For most people, we suggest using a desktop-based application to create a multi-screen video. Conclusion In this article, we have reviewed the top applications for every major platform to create sensational 4-way split-screen videos. Whether you are a professional or a casual user, we wish you could benefit from the knowledge we collected and shared to make your perfect multi-screen video. Multi-window mode enables multiple apps to share the same screen simultaneously. Apps can be side by side or one above the other (split-screen mode), one app in a small window overlaying other apps (picture-in-picture mode), or individual apps in separate movable, resizable windows (free-form mode). Your browser doesn't support the video tag. Figure 1. Display two apps side by side in split-screen mode. The user experience depends on the version of the Android OS and the type of device: Android 7.0 introduces split-screen mode on small screen devices and picture-in-picture mode on select devices. Split-screen mode fills the screen with two apps, showing them either side by side or one above the other. Users can drag the divider separating the two apps to make one larger and the other smaller. Picture-in-picture mode enables users to continue video playback while interacting with another app (see Picture-in-picture support). Manufacturers of large screen devices can enable free-form mode, in which users can freely resize each activity. You can configure how your app handles multi-window mode by specifying your activity's minimum allowable dimensions. You can also disable multi-window mode for your app by setting resizeableActivity="false" to ensure the system always shows your app full screen. Android 8.0 extends picture-in-picture mode to small screen devices. Android 12 makes multi-window mode standard behavior. On large screens (sw >= 600dp), the platform supports all apps in multi-window mode regardless of app configuration. If resizeableActivity="false", the app is put into compatibility mode when necessary to accommodate display dimensions. On small screens (sw < 600dp), the system checks an activity’s minWidth and minHeight to determine whether the activity can run in multi-window mode. If resizeableActivity="false", the app is prevented from running in multi-window mode regardless of minimum width and height. Note: Device manufacturers can override these behaviors. Split-screen mode Users can activate split-screen mode by doing the following: Open the Recents screen Swipe an app into view Press the app icon in the app title bar Select the split screen menu option Select another app from the Recents screen, or close the Recents screen and run another app Users can exit split-screen mode by dragging the window divider to the edge of the screen—up or down, left or right. Note: Android 12L (API level 32) and higher enable users to activate split-screen mode from the Recents screen by selecting the Split action displayed below the active app when two or more apps are in Recents. Launch adjacent If your app needs to access content through an intent, you can use FLAG_ACTIVITY_LAUNCH_ADJACENT to open the content in an adjacent split-screen window. FLAG_ACTIVITY_LAUNCH_ADJACENT was introduced in Android 7.0 (API level 24) to enable apps to start activities in the adjacent window when the device is already in split-screen mode. On Android 12L (API level 32) and higher, the flag enables apps to activate split-screen mode and start activities in an adjacent window. To launch an adjacent activity, use FLAG_ACTIVITY_LAUNCH_ADJACENT in conjunction with FLAG_ACTIVITY_NEW_TASK, for example: fun openUrlInAdjacentWindow(url: String) { Intent(Intent.ACTION_VIEW).apply { data = Uri.parse(url) addFlags(Intent.FLAG_ACTIVITY_LAUNCH_ADJACENT or Intent.FLAG_ACTIVITY_NEW_TASK) }.also { intent -> startActivity(intent) } } public void openUrlInAdjacentWindow(String url) { Intent intent = new Intent(Intent.ACTION_VIEW); intent.setData(Uri.parse(url)); intent.addFlags(Intent.FLAG_ACTIVITY_LAUNCH_ADJACENT | Intent.FLAG_ACTIVITY_NEW_TASK); startActivity(intent); } Note: OEMs can enable 12L behavior on older Android versions, in which case FLAG_ACTIVITY_LAUNCH_ADJACENT functions as it does on API level 32. Activity lifecycle in multi-window mode Multi-window mode does not change the activity lifecycle. However, the resumed state of apps in multiple windows differs on different versions of Android. Multi-resume Android 10 (API level 29) and higher versions support multi-resume—all activities remain in the RESUMED state when the device is in multi-window mode. An activity can be paused if a transparent activity is on top of the activity or the activity is not focusable, for example, picture-in-picture mode. It’s also possible that no activity has focus at a given time, for example, if the notification drawer is open. The onStop method works as usual; the method is called any time an activity is taken off the screen. Multi-resume is also available on select devices running Android 9. To opt in to multi-resume on Android 9 devices, add the following manifest metadata: To verify that a given device supports this manifest metadata, refer to the device specifications. Android 9 In multi-window mode on Android 9 (API level 28) and lower, only the activity the user has most recently interacted with is active at a given time. This activity is considered topmost, and is the only activity in the RESUMED state. All other visible activities are STARTED but are not RESUMED. However, the system gives these visible but not resumed activities higher priority than activities that are not visible. If the user interacts with one of the visible activities, that activity is resumed, and the previously topmost activity enters the STARTED state. When there are multiple activities within a single active app process, the activity with the highest z-order is resumed, and the others are paused. Note: In multi-window mode on Android 9 and lower versions, an app might not be in the RESUMED state even though it is visible to the user, but the app might need to continue its operation while it is not topmost. For example, a video app in this state should continue playing its video. For this reason, we recommend that activities that play video not pause video playback in response to the ON_PAUSE lifecycle event. Instead, the activity should begin playback in response to ON_START, and pause playback in response to ON_STOP. If you handle the lifecycle events directly instead of using the Lifecycle package, pause video playback in your onStop() handler, and resume playback in onStart(). Configuration changes When the user puts an app into multi-window mode, the system notifies the activity of a configuration change as specified in Handle configuration changes. This also happens when the user resizes the app or puts the app back into full screen mode. Essentially, this change has the same activity lifecycle implications as when the system notifies the app that the device has switched from portrait to landscape orientation, except that the device dimensions are changed instead of just being swapped. As discussed in Handle configuration changes, your activity can handle the configuration change itself, or it can allow the system to destroy the activity and recreate it with the new dimensions. If the user is resizing a window and makes it larger in either dimension, the system resizes the activity to match the user action and issues configuration changes as needed. If the app lags behind in drawing in newly exposed areas, the system temporarily fills those areas with the color specified by the windowBackground attribute or by the default windowBackgroundFallback style attribute. Exclusive resource access To help support the multi-resume feature, there’s a new lifecycle callback, onTopResumedActivityChanged(). This method is invoked when an activity gains or loses the top resumed activity position. This is important to know when an activity uses a shared singleton resource, such as the microphone or camera. override fun onTopResumedActivityChanged(topResumed: Boolean) { if (topResumed) { // Top resumed activity // Can be a signal to re-acquire exclusive resources } else { // No longer the top resumed activity } } @Override public void onTopResumedActivityChanged(boolean topResumed) { if (topResumed) { // Top resumed activity // Can be a signal to re-acquire exclusive resources } else { // No longer the top resumed activity } } Note that an app can lose resources for other reasons, such as removal of a shared piece of hardware. In any case, an app should gracefully handle events and state changes that affect available resources. For apps that use a camera, CameraManager.AvailabilityCallback#onCameraAccessPrioritiesChanged() provides a hint that it might be a good time to try to get access to the camera. This method is available as of Android 10 (API level 29). Remember that resizeableActivity=false is not a guarantee of exclusive camera access, since other apps that use the camera can be opened on other displays. Figure 2. Camera in multi-window mode. Your app does not necessary have to release the camera when the app loses focus. For example, you might want to continue camera preview while the user interacts with the newly focused topmost resumed app. It's fine for your app to keep running the camera when it’s not the topmost resumed app, but it has to handle the disconnect case properly. When the topmost resumed app wants to use the camera, it can open it, and your app will lose access. Your app can reopen the camera when the app gets the focus back. After an app receives a CameraDevice.StateCallback#onDisconnected() callback, subsequent calls on the camera device will throw a CameraAccessException. Multi-display Android 10 (API level 29) supports activities on secondary displays. If an activity is running on a device with multiple displays, users can move the activity from one display to another. Multi-resume applies to multi-screen scenarios as well; several activities can receive user input at the same time. An app can specify which display it should run on when it launches or when it creates another activity. This behavior depends on the activity launch mode defined in the manifest file and on the intent flags and options set by the entity launching the activity. See ActivityOptions for more details. When an activity moves to a secondary display, it can go through a context update, window resizing, and configuration and resource changes. If the activity handles the configuration change, the activity is notified in onConfigurationChanged(); otherwise, the activity is relaunched. An activity should check the current display in onCreate and onConfigurationChanged if handling the configuration change. Make sure to update the resources and layouts when the display changes. If the selected launch mode for an activity allows multiple instances, launching on a secondary screen can create a new instance of the activity. Both activities will be resumed at the same time. Figure 3. Multiple instances of an activity on multiple displays. You may also want to read about the multi-display APIs that were introduced in Android 8.0. Activity vs application context Using the right context is crucial in multi-display. When accessing resources, the activity context (which is displayed) is different from the application context (which is not). The activity context contains information about the display and is always adjusted for the display area in which the activity appears. This enables you to get the correct information about the display density or window metrics your app currently has. You should always be using the activity context (or another UI-based context) to get information about the

current window or display. This also affects some system APIs that use information from the context (for example, see Toasts overview). The activity window configuration and parent display define resources and context. Get the current display as follows: val activityDisplay = activity.getDisplay() Display activityDisplay = activity.getDisplay(); Get the current activity window metrics: val windowMetrics = activity.getWindowManager().getCurrentWindowMetrics() WindowMetrics windowMetrics = activity.getWindowManager().getCurrentWindowMetrics(); Get the maximum window metrics for the current system configuration: val maximumWindowMetrics = activity.getWindowManager().getMaximumWindowMetrics() WindowMetrics maximumWindowMetrics = activity.getWindowManager().getMaximumWindowMetrics(); The maximum window metrics are for making calculations, layout choices, or determining the size of resources to fetch ahead of time. Having this available in onCreate() enables you to make these decisions before the first layout pass. These metrics should not be used for laying out specific view elements; instead use information from the Configuration object. Display cutouts Foldable devices might have different cutout geometry when folded and unfolded. To avoid cutout issues read Best practices for display cutout support. You can get the available displays from the DisplayManager system service: val displayManager = getSystemService(Context.DISPLAY_SERVICE) as DisplayManager val displays = displayManager.getDisplays() DisplayManager displayManager = (DisplayManager) getSystemService(Context.DISPLAY_SERVICE); Display[] displays = displayManager.getDisplays(); Use the Display class to get information about a particular display, such as the display size or flags that indicate whether a display is secure. However, do not assume that the display size is going to be the same as the display area allocated to your application. Remember that in multi-window mode, your application occupies a portion of the display. Determine whether an activity can launch on a display: val activityManager = getSystemService(Context.ACTIVITY_SERVICE) as ActivityManager val activityAllowed = activityManager.isActivityStartAllowedOnDisplay(context, displayId, intent) ActivityManager activityManager = (ActivityManager) getSystemService(Context.ACTIVITY_SERVICE); boolean activityAllowed = activityManager.isActivityStartAllowedOnDisplay(context, displayId, intent); Then launch the activity on the display: val options = ActivityOptions.makeBasic() options.setLaunchDisplayId(targetDisplay.displayId) startActivity(intent, options.toBundle()) ActivityOptions options = ActivityOptions.makeBasic(); options.setLaunchDisplayId(targetDisplay.displayId); startActivity(intent, options.toBundle()); Multi-display support Android provides multi-display support for software keyboards, wallpapers, and launchers. Software keyboard A keyboard can be shown on a secondary screen if the display is configured to support system decorations. The input method editor automatically appears if a text field requests input on that display. Figure 4. Keyboard on a secondary display. Wallpaper In Android 10 (API level 29), secondary screens can have wallpaper. The framework creates a separate instance of WallpaperService.Engine for each display. Make sure the surface of each engine is drawn independently. Developers can load assets using the display context in WallpaperService.Engine#getDisplayContext(). Also, make sure your WallpaperInfo.xml file sets android:supportsMultipleDisplays="true". Figure 5. Wallpaper on phone and secondary display. Launchers A new intent filter category, SECONDARY_HOME, provides a dedicated activity for secondary screens. Instances of the activity are used on all displays that support system decorations, one per each display. … … The activity must have a launch mode that does not prevent multiple instances and that can adapt to different screen sizes. The launch mode cannot be singleInstance or singleTask. For example, the AOSP implementation of Launcher3 supports a SECONDARY_HOME activity. Figure 6. Material Design launcher on a phone. Figure 7. Material Design launcher on a secondary display. Window metrics Android 11 (API level 30) introduced the following WindowManager methods to provide the bounds of apps running in multi-window mode: The Jetpack WindowManager library methods computeCurrentWindowMetrics() and computeMaximumWindowMetrics() offer similar functionality respectively, but with backward compatibility to API level 14. To obtain metrics for displays other than the current display, do the following: Create a display context Create a window context for the display Get the WindowManager of the window context Get the WindowMetrics of the maximum display area available to the app val windowMetrics = context.createDisplayContext(display) .createWindowContext(WindowManager.LayoutParams.TYPE_APPLICATION, null) .getSystemService(WindowManager::class.java) .maximumWindowMetrics WindowMetrics windowMetrics = context.createDisplayContext(display) .createWindowContext(WindowManager.LayoutParams.TYPE_APPLICATION, null) .getSystemService(WindowManager.class) .getMaximumWindowMetrics(); Deprecated methods Display methods getSize() and getMetrics() were deprecated in API level 30 in favor of the new WindowManager methods. Android 12 (API level 31) deprecates Display methods getRealSize() and getRealMetrics() and updates their behavior to more closely match the behavior of getMaximumWindowMetrics(). Note: Use Configuration#densityDpi instead of getMetrics() or getRealMetrics() to get the display density. Multi-window mode configuration If your app targets Android 7.0 (API level 24) or higher, you can configure how and whether your app's activities support multi-window mode. You can set attributes in your manifest to control both size and layout. A root activity's attribute settings apply to all activities within its task stack. For example, if the root activity has android:resizeableActivity="true", then all activities in the task stack are resizable. On some larger devices, such as Chromebooks, your app might run in a resizable window even if you specify android:resizeableActivity="false". If this will break your app, you can use filters to restrict your app's availability on such devices. Note: If you build a multi-orientation app that targets API level 23 or lower, and the user uses the app in multi-window mode, the system forcibly resizes the app. The system presents a dialog box warning the user that the app might behave unexpectedly. The system does not resize fixed-orientation apps; if the user attempts to open a fixed-orientation app under multi-window mode, the app takes over the whole screen. Android 12 (API level 31) defaults to multi-window mode. On large screens (sw >= 600dp), all apps run in multi-window mode regardless of app configuration. On small screens, the system checks an activity’s minWidth, minHeight, and resizeableActivity settings to determine whether the activity can run in multi-window mode. resizeableActivity Set this attribute in your manifest's or element to enable or disable multi-window mode for API level 30 and lower: If this attribute is set to true, the activity can be launched in split-screen and free-form modes. If the attribute is set to false, the activity does not support multi-window mode. If this value is false, and the user attempts to launch the activity in multi-window mode, the activity takes over the full screen. If your app targets API level 24 or higher, but you do not specify a value for this attribute, the attribute's value defaults to true. If your app targets API level 31 or higher, this attribute works differently on small and large screens: Large screens (sw >= 600dp): All apps support multi-window mode. The attribute indicates whether an activity can be resized. If resizeableActivity="false", the app is put into compatibility mode when necessary to conform to display dimensions. Small screens (sw < 600dp): If resizeableActivity="true" and activity minimum width and minimum height are within the multi-window requirements, the activity supports multi-window mode. If resizeableActivity="false", the activity does not support multi-window mode regardless of activity minimum width and height. supportsPictureInPicture Set this attribute in your manifest's node to indicate whether the activity supports picture-in-picture mode. Note: If supportsPictureInPicture="true", you must set the android:configChanges attribute to enable your activity to handle configuration changes (see Picture-in-picture support). configChanges To handle multi-window configuration changes yourself, such as when a user resizes a window, add the android:configChanges attribute to your app manifest node with at least the following values: After adding android:configChanges, your activity and fragments receive a callback to onConfigurationChanged() instead of being destroyed and recreated. You can then manually update your views, reload resources, and perform other operations as needed. With Android 7.0, the manifest element supports several attributes that affect how an activity behaves in multi-window mode: android:defaultWidth Default width of the activity when launched in free-form mode. android:defaultHeight Default height of the activity when launched in free-form mode. android:gravity Initial placement of the activity when launched in free-form mode. See the Gravity reference for suitable values. android:minHeight, android:minWidth Minimum height and minimum width for the activity in both split-screen and free-form modes. If the user moves the divider in split-screen mode to make an activity smaller than the specified minimum, the system crops the activity to the size the user requests. The following code shows how to specify an activity's default size and location and its minimum size when the activity is displayed in free-form mode: Multi-window mode at runtime Beginning with Android 7.0, the system offers functionality to support apps that can run in multi-window mode. Disabled features in multi-window mode In multi-window mode, Android might disable or ignore features that don't apply to an activity that is sharing the device screen with other activities or apps. Additionally, some system UI customization options are disabled. For example, apps cannot hide the status bar if they are running in multi-window mode (see Control the system UI visibility). The system ignores changes to the android:screenOrientation attribute. Multi-window mode queries and callbacks The Activity class offers the following methods to support multi-window mode: isInMultiWindowMode() Indicates whether the activity is in multi-window mode. isInPictureInPictureMode() Indicates whether the activity is in picture-in-picture mode. Note: Picture-in-picture mode is a special case of multi-window mode. If myActivity.isInPictureInPictureMode() returns true, then myActivity.isInMultiWindowMode() also returns true. onMultiWindowModeChanged() The system calls this method whenever the activity goes into or out of multi-window mode. The system passes the method a value of true if the activity is entering multi-window mode or false if the activity is leaving multi-window mode. onPictureInPictureModeChanged() The system calls this method whenever the activity goes into or out of picture-in-picture mode. The system passes the method a value of true if the activity is entering picture-in-picture mode or false if the activity is leaving picture-in-picture mode. The Fragment class exposes versions of many of these methods; for example, Fragment.onMultiWindowModeChanged(). Picture-in-picture mode To put an activity in picture-in-picture mode, call enterPictureInPictureMode() This method has no effect if the device does not support picture-in-picture mode. For more information, see Picture-in-picture support. New activities in multi-window mode When you launch a new activity, you can indicate that the new activity should be displayed adjacent to the current one if possible. Use the intent flag FLAG_ACTIVITY_LAUNCH_ADJACENT, which tells the system to try to create the new activity in an adjacent window, so the two activities share the screen. The system makes a best effort to do this, but it is not guaranteed to happen. If a device is in free-form mode and you are launching a new activity, you can specify the new activity's dimensions and screen location by calling ActivityOptions.setLaunchBounds(). This method has no effect if the device is not in multi-window mode. On API level 30 and lower, if you launch an activity within a task stack, the activity replaces the activity on the screen, inheriting all of its multi-window properties. If you want to launch the new activity as a separate window in multi-window mode, you must launch it in a new task stack. Android 12 (API level 31) enables apps to split an application's task window among multiple activities. You determine how your app displays its activities — full screen, side by side, or stacked — by creating an XML configuration file or making Jetpack WindowManager API calls. Drag and drop Users can drag and drop data from one activity to another while the two activities are sharing the screen. (Prior to Android 7.0, users could only drag and drop data within a single activity.) To quickly add support for accepting dropped content see the DropHelper API. For comprehensive drag-and-drop guidance, see Drag and drop. Multi-instance Each root activity has its own task, which runs on a separate process and is displayed in its own window. To launch a new instance of your app in a separate window, you can start new activities with the FLAG_ACTIVITY_NEW_TASK flag. You can combine this with some of the multi-window attributes to request a specific location for the new window. For example, a shopping app can display multiple windows to compare products. Android 12 (API level 31) enables you to launch two instances of an activity side by side in the same task window. If you want to allow users to start another instance of your application from the application launcher or the taskbar, make sure that your launcher Activity sets android:resizeableActivity="true" and does not use a launch mode that prevents multiple instances. For example a singleInstancePerTask activity can be instantiated multiple times in different tasks when FLAG_ACTIVITY_MULTIPLE_TASK or FLAG_ACTIVITY_NEW_DOCUMENT is set. Note: The application launcher is a system dialog that displays a list of apps that satisfy a specified intent. Users can launch an app by selecting it from the list. See Intent types. Don't confuse multi-instance with a multi-panel layout, such as a list detail layout that uses SlidingPaneLayout, which runs inside a single window. Note that when multiple instances are running in separate windows on a foldable device, one or more instances might be sent to the background if the posture changes. For example, assume a device is unfolded and has two app instances running in two windows on either side of the fold. If the device is folded, one of the instances might be terminated instead of trying to fit the windows for both instances on a smaller screen. Multi-window mode verification Whether or not your app targets API level 24 or higher, you should verify how it behaves in multi-window mode in case a user tries to launch it in multi-window mode on a device running Android 7.0 or higher. Caution: Unity apps running on Unity Long Term Support (LTS) version 2018 or earlier lose focus and the app window turns black when the app is running full screen and split-screen mode is activated. The app can be restored by focusing the app window. To eliminate the issue, upgrade your Unity app to LTS version 2019 or later. Test devices Devices that run Android 7.0 (API level 24) or higher support multi-window mode. API level 23 or lower When users attempt to use the app in multi-window mode, the system forcibly resizes the app unless the app declares a fixed orientation. If your app does not declare a fixed orientation, you should launch your app on a device running Android 7.0 or higher and attempt to put the app in split-screen mode. Verify that the user experience is acceptable when the app is forcibly resized. If the app declares a fixed orientation, you should attempt to put the app in multi-window mode. Verify that when you do so, the app remains in full screen mode. API levels 24 through 30 If your app targets API levels 24 through 30 and does not disable multi-window support, verify the following behavior under both split-screen and free-form modes: Launch the app full screen, then switch to multi-window mode by long-pressing the Recents button. Verify that the app switches properly. Launch the app directly in multi-window mode and verify that the app launches properly. You can launch an app in multi-window mode by pressing the Recents button, then long-pressing the title bar of your app and dragging it to one of the highlighted areas on the screen. Resize your app in split-screen mode by dragging the screen divider. Verify that the app resizes without crashing and that necessary UI elements are visible. If you have specified minimum dimensions for your app, attempt to resize the app below those dimensions. Verify that you cannot resize the app to be smaller than the specified minimum dimensions. Through all tests, verify that your app's performance is acceptable. For example, verify that there is not too long a lag to update the UI after the app is resized. API level 31 or higher If your app targets API level 31 or higher and the main activity's minimum width and minimum height are less than or equal to the respective dimensions of the available display area, verify all the behaviors listed for API levels 24 through 30. Note: You can programmatically determine whether your app is in multi-window mode by checking the return value of Activity#isInMultiWindowMode(). Test checklist To verify your app's performance in multi-window mode, try the following operations. You should try these operations in both split-screen and free-form mode, except where otherwise noted. Enter and leave multi-window mode. Switch from your app to another app, and verify that the app behaves properly while it is visible but not active. For example, if your app is playing video, verify that the video continues to play while the user is interacting with another app. In split-screen mode, try moving the screen divider to make your app both larger and smaller. Try these operations in both side by side and one above the other configurations. Verify that the app does not crash, essential functionality is visible, and the resize operation doesn't take too long. Perform several resize operations in rapid succession. Verify that your app doesn't crash or leak memory. Android Studio's Memory Profiler provides information about your app's memory usage (see Inspect your app's memory usage with Memory Profiler). Use your app normally in a number of different window configurations, and verify that the app behaves properly. Verify that text is readable and that UI elements aren't too small to interact with. Multi-window support disabled On API levels 24 through 30, if you disabled multi-window support by setting android:resizeableActivity="false", you should launch your app on a device running Android 7.0 through 11 and attempt to put the app in split-screen and free-form modes. Verify that when you do so, the app remains in full-screen mode. Additional resources For further information about multi-window support in Android, see: