<activity>

syntax:

```
<activity android:allowEmbedded (#embedded)=["true" | "false"]
          android:allowTaskReparenting (#reparent)=["true" | "false"]
          android:alwaysRetainTaskState (#always)=["true" | "false"]
          android:autoRemoveFromRecents (#autoremrecents)=["true" | "false"]
          android:banner (#banner)="drawable resource"
          android:canDisplayOnRemoteDevices (#canDisplayOnRemoteDevices)=["true"
          android:clearTaskOnLaunch (#clear)=["true" | "false"]
          android:colorMode (#colormode)=[ "hdr" | "wideColorGamut"]
          android:configChanges (#config)=["colorMode", "density",
                                 "fontScale", "fontWeightAdjustment",
                                 "grammaticalGender", "keyboard",
                                 "keyboardHidden", "layoutDirection", "loca
                                 "mcc", "mnc", "navigation", "orientation",
                                 "screenLayout", "screenSize",
                                 "smallestScreenSize", "touchscreen", "uiMo
          android:directBootAware (#directBootAware)=["true" | "false"]
          android:documentLaunchMode (#dlmode)=["intoExisting" | "always" |
                                  "none" | "never"]
          android:enabled (#enabled)=["true" | "false"]
          android:enableOnBackInvokedCallback (#enableOnBackInvokedCallback)=["t
          android:excludeFromRecents (#exclude)=["true" | "false"]
          android:exported (#exported)=["true" | "false"]
          android:finishOnTaskLaunch (#finish)=["true" | "false"]
          android:forceQueryable (#forceQueryable)=["true" | "false"]
          android:hardwareAccelerated (#hwaccel)=["true" | "false"]
          android:icon (#icon)="drawable resource"
          android:immersive (#immersive)=["true" | "false"]
          android:intentMatchingFlags (#intentMatchingFlags)=["none" | "enforceI
          android:label (#label)="string resource"
          android:launchMode (#lmode)=["standard" | "singleTop" |
                              "singleTask" | "singleInstance" | "singleInst
          android:lockTaskMode (#ltmode)=["normal" | "never" |
                              "if_whitelisted" | "always"]
          android:maxRecents (#maxrecents)="integer"
          android:maxAspectRatio (#maxaspectratio)="float"
          android:minAspectRatio (#minaspectratio)="float"
          android:multiprocess (#multi)=["true" | "false"]
          android:name (#nm)="string"
```

```
        android:noHistory (#nohist)=["true" | "false"]
        android:parentActivityName (#parent)="string"
        android:persistableMode (#persistableMode)=["persistRootOnly" |
                                "persistAcrossReboots" | "persistNever"]
        android:permission (#prmsn)="string"
        android:preferMinimalPostProcessing (#preferMinimalPostProcessing)=["tru
        android:process (#proc)="string"
        android:recreateOnConfigChanges (#recreateOnConfigChanges)=["colorMod
                                        "keyboardHidden" | "mcc" | "mnc"
                                        "navigation" | "touchscreen"]
        android:relinquishTaskIdentity (#relinquish)=["true" | "false"]
        android:requireContentUriPermissionFromCaller (#requireContentUriPermis
                                                        "readOrWrite" | "w
        android:resizeableActivity (#resizeableActivity)=["true" | "false"]
        android:screenOrientation (#screen)=["unspecified" | "behind" |
                                    "landscape" | "portrait" |
                                    "reverseLandscape" | "reversePortrait"
                                    "sensorLandscape" | "sensorPortrait" |
                                    "userLandscape" | "userPortrait" |
                                    "sensor" | "fullSensor" | "nosensor" |
                                    "user" | "fullUser" | "locked"]
        android:showForAllUsers (#showForAllUsers)=["true" | "false"]
        android:stateNotNeeded (#state)=["true" | "false"]
        android:supportsPictureInPicture (#supportsPIP)=["true" | "false"]
        android:taskAffinity (#aff)="string"
        android:theme (#theme)="resource or theme"
        android:uiOptions (#uioptions)=["none" | "splitActionBarWhenNarrow"]
        android:windowSoftInputMode (#wsoft)=["stateUnspecified",
                                      "stateUnchanged", "stateHidden",
                                      "stateAlwaysHidden", "stateVisible",
                                      "stateAlwaysVisible", "adjustUnspeci
                                      "adjustResize", "adjustPan"] >
    ...
</activity>
```

contained in:

`<application>` (/guide/topics/manifest/application-element)

can contain:

`<intent-filter>` (/guide/topics/manifest/intent-filter-element)
`<meta-data>` (/guide/topics/manifest/meta-data-element)

`<layout>` (/guide/topics/manifest/layout-element)
`<property>` (/guide/topics/manifest/property-element)

description:

Declares an activity (an `Activity` (/reference/android/app/Activity) subclass) that implements part of the application's visual user interface. All activities must be represented by `<activity>` elements in the manifest file. Any that aren't declared there aren't seen by the system and never run.

attributes:

`android:allowEmbedded`

Indicates that the activity can be launched as the embedded child of another activity, particularly in the case where the child lives in a container, such as a `Display` owned by another activity. For example, activities that are used for Wear custom notifications declare this so Wear can display the activity in its context stream, which resides in another process.

The default value of this attribute is `false`.

`android:allowTaskReparenting`

Whether the activity can move from the task that started it to the task it has an affinity for when that task is next brought to the front. It's `"true"` if it can move, and `"false"` if it remains with the task where it started.

If this attribute isn't set, the value set by the corresponding `allowTaskReparenting` (/guide/topics/manifest/application-element#reparent) attribute of the `<application>` (/guide/topics/manifest/application-element) element applies to the activity. The default value is `"false"`.

Normally, when an activity is started it's associated with the task of the activity that started it and it stays there for its entire lifetime. You can use this attribute to force it to be re-parented to the task it has an affinity for when its current task is no longer displayed. Typically, this is used to cause the activities of an application to move to the main task associated with that application.

For example, if an email message contains a link to a web page, clicking the link brings up an activity that can display the page. That activity is defined by the

browser application but is launched as part of the email task. If it's reparented to the browser task, it shows when the browser next comes to the front and is absent when the email task again comes forward.

The affinity of an activity is defined by the `taskAffinity` (#aff) attribute. The affinity of a task is determined by reading the affinity of its root activity. Therefore, by definition, a root activity is always in a task with the same affinity. Since activities with `"singleTask"` or `"singleInstance"` launch modes can only be at the root of a task, re-parenting is limited to the `"standard"` and `"singleTop"` modes. (See also the `launchMode` (#lmode) attribute.)

`android:alwaysRetainTaskState`

Whether the state of the task that the activity is in is always maintained by the system. `"true"` if it is, and `"false"` if the system can reset the task to its initial state in certain situations. The default value is `"false"`. This attribute is meaningful only for the root activity of a task. It's ignored for all other activities.

Normally, the system clears a task, removing all activities from the stack above the root activity, in certain situations when the user re-selects that task from the home screen. Typically, this is done if the user hasn't visited the task for a certain amount of time, such as 30 minutes.

However, when this attribute is `"true"`, users always return to the task in its last state, regardless of how they get there. This is useful in an application like a web browser where there is a lot of state, such as multiple open tabs, that users don't want to lose.

`android:autoRemoveFromRecents`

Whether tasks launched by the activity with this attribute remain in the Recents screen (/guide/components/recents) until the last activity in the task completes. If `true`, the task is automatically removed from the Recents screen. This overrides the caller's use of `FLAG_ACTIVITY_RETAIN_IN_RECENTS` (/reference/android/content/Intent#FLAG_ACTIVITY_RETAIN_IN_RECENTS). It must be a boolean value, either `"true"` or `"false"`.

`android:banner`

A drawable resource (/guide/topics/resources/drawable-resource) providing an extended graphical banner for its associated item. Use with the `<activity>` tag

to supply a default banner for a specific activity or with the `<application>` (/guide/topics/manifest/application-element) tag to supply a banner for all application activities.

The system uses the banner to represent an app in the Android TV home screen. Since the banner displays only in the home screen, it is only specified by applications with an activity that handles the `CATEGORY_LEANBACK_LAUNCHER` (/reference/android/content/Intent#CATEGORY_LEANBACK_LAUNCHER) intent.

This attribute is set as a reference to a drawable resource containing the image, such as `"@drawable/banner"`. There is no default banner.

For more information, see Provide a home screen banner (/training/tv/start/start#banner) in Get Started with TV Apps.

`android:canDisplayOnRemoteDevices`

Indicates whether the activity can be displayed on a remote device which may or may not be running Android. It must be a boolean value, either `"true"` or `"false"`.

The default value of this attribute is `"true"`.

`android:clearTaskOnLaunch`

Whether all activities are removed from the task, except for the root activity, when it is re-launched from the home screen. `"true"` if the task is always stripped down to its root activity, and `"false"` if not. The default value is `"false"`. This attribute is meaningful only for activities that start a new task—the root activity. It's ignored for all other activities in the task.

When the value is `"true"`, every time users start the task, they are brought to its root activity regardless of what they were last doing in the task and regardless of whether they used the Back or Home button to leave it. When the value is `"false"`, the task can be cleared of activities in some situations, but not always. For more information, see the `alwaysRetainTaskState` (#always) attribute.

Suppose the user launches activity P from the home screen, and from there goes to activity Q. The user next taps Home, and then returns to activity P. Normally, the user sees activity Q, since that is what they were last doing in P's task. However, if P set this flag to `"true"`, all of the activities on top of it—in this case,

Q—are removed when the user launches activity P from the home screen. So, the user sees only P when returning to the task.

If this attribute and `allowTaskReparenting` (#reparent) are both `"true"`, any activities that can be re-parented are moved to the task they share an affinity with. The remaining activities are then dropped.

This attribute is ignored if `FLAG_ACTIVITY_RESET_TASK_IF_NEEDED` (/reference/android/content/Intent#FLAG_ACTIVITY_RESET_TASK_IF_NEEDED) isn't set.

`android:colorMode`

Specifies the activity's color mode. If specified, can be either `hdr` or `wideColorGamut`.

If `hdr`, requests that the activity be displayed in a high dynamic range if the device supports it.

★ **Note:** For performance reasons, we do not recommend turning on HDR in the manifest. Instead, if your app is displaying HDR images, you should call `setColorMode()` (/reference/android/view/Window#setColorMode(int)) to dynamically switch the activity to HDR mode at run time. For more information, see Display Ultra HDR images (/media/grow/ultra-hdr-display).

If `wideColorGamut`, requests that the activity be displayed in wide color gamut mode on compatible devices. In wide color gamut mode, a window can render outside of the `SRGB` (/reference/android/graphics/ColorSpace.Named#SRGB) gamut to display more vibrant colors. If the device doesn't support wide color gamut rendering, this attribute has no effect. For more information about rendering in wide color mode, see Enhance graphics with wide color content (/training/wide-color-gamut).

`android:configChanges`

Lists configuration changes that the activity handles itself. When a configuration change occurs at runtime, the activity shuts down and restarts by default, but declaring a configuration with this attribute prevents the activity from restarting. Instead, the activity remains running and its `onConfigurationChanged()`

(/reference/android/app/Activity#onConfigurationChanged(android.content.res.Configuration))
method is called.

★ **Note:** Use this attribute only in special cases to improve application performance and responsiveness. For more information, see Handle configuration changes (/guide/topics/resources/runtime-changes).

The following strings are valid values for this attribute. Multiple values are separated by `|`, such as `"locale|navigation|orientation"`.

Value	Description
`"colorMode"`	The color mode capabilities of the screen (color gamut or dynamic range) have changed. ★ **Note:** The color mode the activity requests with the **`colorMode`** attribute or at runtime is distinct from the capability for different color modes. An activity changing the color mode it is using does not cause a configuration change, because the color capabilities of the display have not changed.
`"density"`	A change to the display density, such as when the user specifies a different display scale or a different display is now active. *Added in API level 24*.
`"fontScale"`	A change to the font scaling factor, such as when the user selects a new global font size.
`"fontWeightAdjustment"`	The amount of the font weight increase has changed.
`"grammaticalGender"`	The grammatical gender of the language has changed. See `GrammaticalInflectionManager`

	(/reference/kotlin/android/app/GrammaticalInflectionManager). *Added in API level 34*.
`"keyboard"`	A change to the keyboard type, such as when the user plugs in an external keyboard.
`"keyboardHidden"`	A change to the keyboard accessibility, such as when the user reveals the hardware keyboard.
`"layoutDirection"`	A change to the layout direction, such as from left-to-right (LTR) to right-to-left (RTL). *Added in API level 17*.
`"locale"`	A change to the locale, such as when the user selects a new language that text displays in.
`"mcc"`	A change to the IMSI mobile country code (MCC) when a SIM is detected that updates the MCC.
`"mnc"`	A change to the IMSI mobile network code (MNC) when a SIM is detected that updates the MNC.
`"navigation"`	TA change to the navigation type (trackball or D-pad). Normally, this does not happen.
`"orientation"`	A change to the screen orientation, such as when the user rotates the device. ★ **Note:** If your application targets Android 3.2 (API level 13) or higher, also declare the `"screenLayout"` and `"screenSize"` configurations, because screen layout and screen size can change when a device switches between portrait and landscape orientations.
`"screenLayout"`	A change to the screen layout, such as when a different display becomes active.

`"screenSize"`	A change to the current available screen size. This represents a change in the currently available size, relative to the current aspect ratio, so it changes when the user switches between landscape and portrait. *Added in API level 13*.
`"smallestScreenSize"`	A change to the physical screen size. This represents a change in size regardless of orientation, so it only changes when the actual physical screen size changes, such as switching to an external display. A change to this configuration corresponds to a change in the `smallestWidth` configuration (/guide/topics/resources/providing-resources#SmallestScreenWidthQualifier). *Added in API level 13*.
`"touchscreen"`	A change to touchscreen mode, such as when the user connects or disconnects an input peripheral or moves the app between different displays.
`"uiMode"`	A change to the user interface mode, such as when the user places the device into a desk or car dock, or the night mode changes. For more information about the different UI modes, see `UiModeManager` (/reference/android/app/UiModeManager). *Added in API level 8*.

All these configuration changes can impact the resource values seen by the application. Therefore, when `onConfigurationChanged()` (/reference/android/app/Activity#onConfigurationChanged(android.content.res.Configuration)) is called, it is usually necessary to again retrieve all resources, including view layouts and drawables, to correctly handle the change.

★ **Note:** To handle multi-window (/guide/topics/ui/multi-window) related configuration changes, use both `"screenLayout"` and `"smallestScreenSize"`. Multi-window is supported in Android 7.0 (API level 24) or higher.

`android:directBootAware`

Whether the activity is *Direct-Boot aware*—that is, whether it can run before the user unlocks the device.

★ **Note:** During Direct Boot (/training/articles/direct-boot), an activity in your application can only access the data that is stored in *device protected* storage.

The default value is `"false"`.

`android:documentLaunchMode`

Specifies how a new instance of an activity is added to a task each time it is launched. This attribute permits the user to have multiple documents from the same application appear in the Recents screen (/guide/components/recents).

This attribute has four values, which produce the following effects when the user opens a document with the application:

Value	Description
`"intoExisting"`	The system searches for a task whose base intent's `ComponentName` and data URI match those of the launching intent. If the system finds such a task, the system clears the task and restarts, with the root activity receiving a call to `onNewIntent(android.content.Intent)` (/reference/android/app/Activity#onNewIntent(android.content.Intent)). If the system doesn't find such a task, the system creates a new task.
`"always"`	The activity creates a new task for the document, even if the document is already opened. This is the same as setting both the `FLAG_ACTIVITY_NEW_DOCUMENT` (/reference/android/content/Intent#FLAG_ACTIVITY_NEW_DOCUMENT) and

`FLAG_ACTIVITY_MULTIPLE_TASK` (/reference/android/content/Intent#FLAG_ACTIVITY_MULTIPLE_TASK) flags.

`"none"`	The activity doesn't create a new task for the activity. This is the default value, which creates a new task only when `FLAG_ACTIVITY_NEW_TASK` (/reference/android/content/Intent#FLAG_ACTIVITY_NEW_TASK) is set. The Recents screen treats the activity as it does by default: it displays a single task for the app, which resumes from whatever activity the user last invoked.
`"never"`	The activity isn't launched into a new document even if the intent contains `FLAG_ACTIVITY_NEW_DOCUMENT` (/reference/android/content/Intent#FLAG_ACTIVITY_NEW_DOCUMENT). Setting this overrides the behavior of the `FLAG_ACTIVITY_NEW_DOCUMENT` and `FLAG_ACTIVITY_MULTIPLE_TASK` (/reference/android/content/Intent#FLAG_ACTIVITY_MULTIPLE_TASK) flags, if either of these are set in the activity, and the Recents screen displays a single task for the app, which resumes from whatever activity the user last invoked.

★ **Note:** For values other than `"none"` and `"never"`, the activity is defined with `launchMode="standard"`. If this attribute isn't specified, `documentLaunchMode="none"` is used.

`android:enabled`

Whether the activity can be instantiated by the system. It's `"true"` if it can be, and `"false"` if not. The default value is `"true"`.

The `<application>` (/guide/topics/manifest/application-element) element has its own `enabled` (/guide/topics/manifest/application-element#enabled) attribute that applies to all application components, including activities. The `<application>` and `<activity>` attributes must both be `"true"`, as they both are by default, for the system to be able to instantiate the activity. If either is `"false"`, it can't be instantiated.

`android:enableOnBackInvokedCallback`

This flag lets you opt out of predictive system animations at the activity level.

Set `android:enableOnBackInvokedCallback=false` to turn off predictive back animations at the activity level and instruct the system to ignore calls to the `OnBackInvokedCallback` platform API.

`android:excludeFromRecents`

Whether the task initiated by this activity is excluded from the Recents screen (/guide/components/recents). That is, when this activity is the root activity of a new task, this attribute determines whether the task appears in the list of recent apps. It's `"true"` if the task is *excluded* from the list; `"false"` if it is *included*. The default value is `"false"`.

`android:exported`

Whether the activity can be launched by components of other applications:

- If `"true"`, the activity is accessible to any app, and is launchable by its exact class name.
- If `"false"`, the activity can be launched only by components of the same application, applications with the same user ID, or privileged system components. This is the default value when there are no intent filters.

If an activity in your app includes intent filters, set this element to `"true"` to let other apps start it. For example, if the activity is the main activity of the app and includes the `category` (/guide/topics/manifest/category-element) `android.intent.category.LAUNCHER` (/reference/android/content/Intent#CATEGORY_LAUNCHER).

If this element is set to `"false"` and an app tries to start the activity, the system throws an `ActivityNotFoundException` (/reference/android/content/ActivityNotFoundException).

This attribute isn't the only way to limit an activity's exposure to other applications. Permissions are also used to limit the external entities that can invoke the activity. See the `permission` (/guide/topics/manifest/activity-element#prmsn) attribute.

`android:finishOnTaskLaunch`

Whether an existing instance of the activity is shut down, except for the root activity, when the user re-launches its task by choosing the task on the home screen. It's `"true"` if it is shut down, and `"false"` if not. The default value is `"false"`.

If this attribute and `allowTaskReparenting` (/guide/topics/manifest/activity-element#reparent) are both `"true"`, this attribute trumps the other. The affinity of the activity is ignored. The activity isn't re-parented, but destroyed.

This attribute is ignored if `FLAG_ACTIVITY_RESET_TASK_IF_NEEDED` (/reference/android/content/Intent#FLAG_ACTIVITY_RESET_TASK_IF_NEEDED) isn't set.

`android:forceQueryable`

Specifies whether this activity is visible to all other applications on the device, bypassing normal package visibility restrictions.

The default value is `"false"`.

For more information, see the guide on automatic package visibility filtering (/training/package-visibility/automatic).

`android:hardwareAccelerated`

Whether hardware-accelerated rendering is enabled for this activity. `"true"` if it is enabled, and `"false"` if not. The default value is `"false"`.

On Android 3.0 and higher, a hardware-accelerated OpenGL renderer is available to applications to improve performance for many common 2D graphics operations. When the hardware-accelerated renderer is enabled, most operations in Canvas, Paint, Xfermode, ColorFilter, Shader, and Camera are accelerated.

This results in smoother animations, smoother scrolling, and improved responsiveness overall, even for applications that don't explicitly make use the framework's OpenGL libraries. Because of the increased resources required to enable hardware acceleration, your app consumes more RAM.

Not all of the OpenGL 2D operations are accelerated. If you enable the hardware-accelerated renderer, test whether your application can use the renderer without errors.

`android:icon`

An icon representing the activity. The icon is displayed to users when a representation of the activity is required on-screen. For example, icons for activities that initiate tasks are displayed in the launcher window. The icon is often accompanied by a label; for information about the label, see the `android:label` (#label) attribute.

This attribute is set as a reference to a drawable resource containing the image definition. If it isn't set, the icon specified for the application as a whole is used instead. For more information, see the `<application>` (/guide/topics/manifest/application-element) element's `icon` (/guide/topics/manifest/application-element#icon) attribute.

The activity's icon, whether set here or by the `<application>` element, is also the default icon for all the activity's intent filters. For more information, see the `<intent-filter>` (/guide/topics/manifest/intent-filter-element) element's `icon` (/guide/topics/manifest/intent-filter-element#icon) attribute.

`android:immersive`

Sets the immersive mode setting for the current activity. If it's `"true"`, the `ActivityInfo.flags` (/reference/android/content/pm/ActivityInfo#flags) member always has its `FLAG_IMMERSIVE` (/reference/android/content/pm/ActivityInfo#FLAG_IMMERSIVE) bit set, even if the immersive mode changes at runtime using the `setImmersive()` (/reference/android/app/Activity#setImmersive(boolean)) method.

`android:intentMatchingFlags`

Use this attribute to fine-tune how the system matches incoming intents to app components. By default, no special matching rules are applied.

The value set on an `<activity>` tag overrides the value set on the `<application>` tag.

The value must be one or more of the following flags, separated by '`|`':

Flag	Description

`none`	Disables all special matching rules for incoming intents. When specifying multiple flags, conflicting values are resolved by giving precedence to the `none` flag.
`enforceIntentFilter`	Enforces stricter matching for incoming intents: • Explicit intents must match the target component's intent filter. • Intents without an action don't match any intent filter.
`allowNullAction`	Relaxes the matching rules to allow intents without an action to match. This flag is used in conjunction with `enforceIntentFilter` to achieve the following behavior: • Explicit intents must match the target component's intent filter. • Intents without an action are allowed to match any intent filter.

For more information, see the Safer Intents (/about/versions/16/behavior-changes-16#safer-intents) section in the Android 16 (API level 36) behavior changes.

`android:label`

A user-readable label for the activity. The label displays on-screen when the activity is represented to the user. It's often displayed along with the activity icon. If this attribute isn't set, the label set for the application as a whole is used instead. see the `<application>` (/guide/topics/manifest/application-element) element's `label` (/guide/topics/manifest/application-element#label) attribute.

The activity's label, whether set here or by the `<application>` element, is also the default label for all the activity's intent filters. For more information, see the `<intent-filter>` (/guide/topics/manifest/intent-filter-element) element's `label` (/guide/topics/manifest/intent-filter-element#label) attribute.

The label is set as a reference to a string resource so that it can be localized like other strings in the user interface. However, as a convenience while you're developing the application, it can also be set as a raw string.

`android:launchMode`

An instruction for how the activity launches. There are five modes, which work in conjunction with activity flags (`FLAG_ACTIVITY_*` constants) in `Intent` (/reference/android/content/Intent) objects to determine what happens when the activity is called upon to handle an intent:

`"standard"`
`"singleTop"`
`"singleTask"`
`"singleInstance"`
`"singleInstancePerTask"`

The default mode is `"standard"`.

As shown in the following table, the modes fall into two main groups, with `"standard"` and `"singleTop"` activities on one side, and `"singleTask"`, `"singleInstance"`, and `"singleInstancePerTask"` activities on the other. An activity with the `"standard"` or `"singleTop"` launch mode can be instantiated multiple times.

The instances can belong to any task and can be located anywhere in the activity task. Typically, they're launched into the task that called `startActivity()` (/reference/android/content/Context#startActivity(android.content.Intent)), unless the `Intent` object contains a `FLAG_ACTIVITY_NEW_TASK` (/reference/android/content/Intent#FLAG_ACTIVITY_NEW_TASK) instruction, in which case a different task is chosen. For more information, see the `taskAffinity` (#aff) attribute.

In contrast, `"singleTask"`, `"singleInstance"`, and `"singleInstancePerTask"` activities have different behaviors. `"singleInstancePerTask"` is always at the root of the activity task. Also, the device can hold only one instance of the `"singleInstance"` activity at a time, while the `"singleInstancePerTask` activity can be instantiated multiple times in different tasks when `FLAG_ACTIVITY_MULTIPLE_TASK` (/reference/android/content/Intent#FLAG_ACTIVITY_MULTIPLE_TASK) or `FLAG_ACTIVITY_NEW_DOCUMENT` (/reference/android/content/Intent#FLAG_ACTIVITY_NEW_DOCUMENT) is set.

An activity with the `"singleTask"` launch mode combines the behaviors of `"singleInstance"` and `"singleInstancePerTask"`: the activity can be instantiated multiple times and can be located anywhere in a task of the same

`taskAffinity`. But the device can only hold one task for locating the `"singleTask"` activity at the root of the activity task.

The `"standard"` and `"singleTop"` modes differ from each other in one respect: every time there's a new intent for a `"standard"` activity, a new instance of the class is created to respond to that intent. Each instance handles a single intent. Similarly, a new instance of a `"singleTop"` activity can also be created to handle a new intent.

However, if the target task already has an existing instance of the activity at the top of its stack, that instance receives the new intent, in an `onNewIntent()` (/reference/android/app/Activity#onNewIntent(android.content.Intent)) call. A new instance isn't created. Otherwise—if an existing instance of the `"singleTop"` activity is in the target task but not at the top of the stack, or if it's at the top of a stack, but not in the target task—a new instance is created and pushed on the stack.

Similarly, if the user navigates up (/training/implementing-navigation/ancestral) to an activity on the current stack, the behavior is determined by the parent activity's launch mode. If the parent activity has launch mode `singleTop` (or the `up` intent contains `FLAG_ACTIVITY_CLEAR_TOP` (/reference/android/content/Intent#FLAG_ACTIVITY_CLEAR_TOP)), the parent is brought to the top of the stack, and its state is preserved.

The navigation intent is received by the parent activity's `onNewIntent()` (/reference/android/app/Activity#onNewIntent(android.content.Intent)) method. If the parent activity has launch mode `standard`, and the `up` intent doesn't contain `FLAG_ACTIVITY_CLEAR_TOP`, the current activity and its parent both pop off the stack, and a new instance of the parent activity is created to receive the navigation intent.

The `"singleInstance"` mode also differs from `"singleTask"` and `"singleInstancePerTask"` in only one respect: an activity with the `"singleTask"` or `"singleInstancePerTask"` launch mode lets other activities, necessarily `"standard"` and `"singleTop"` activities, be part of its task.

A `"singleInstance"` activity, on the other hand, permits no other activities to be part of its task. It must be the only activity in the task. If it starts another activity, that activity is assigned to a different task, as if `FLAG_ACTIVITY_NEW_TASK` were in the intent.

Use cases	Launch mode	Multiple instances?	Comments
Normal launches for most activities	`"standard"`	Yes	Default. The system always creates a new instance o target task and routes the intent to it.
	`"singleTop"`	Conditionally	If an instance of the activity already exists at the top the system routes the intent to that instance through `NewIntent()` (/reference/android/app/Activity#onNewIntent(andro method, rather than creating a new instance of the a
Specialized launches *(not recommended for general use)*	`"singleTask"`	Conditionally	The system creates the activity at the root of a new t activity on an existing task with the same affinity. If a activity already exists and is at the root of the task, th the intent to existing instance through a call to its `on` (/reference/android/app/Activity#onNewIntent(andro method, rather than creating a new one.
	`"singleInstance"`	No	Same as `"singleTask"`, except that the system do other activities into the task holding the instance. Th the single and only member of its task.
	`"singleInstancePerTask"`	Conditionally	The activity can only run as the root activity of the ta activity that created the task, and therefore there is o of this activity in a task. However, the activity can be multiple times in different tasks.

As shown in the preceding table, `"standard"` is the default mode and is appropriate for most types of activities. `"singleTop"` is also a common and useful launch mode for many types of activities. The other modes, `"singleTask"`, `"singleInstance"`, and `"singleInstancePerTask"`, are *not appropriate* for most applications. They result in an interaction model that is likely to be unfamiliar to users and is very different from most other applications.

Regardless of the launch mode that you choose, make sure to test the usability of the activity during launch and when navigating back to it from other activities and tasks using the Back button.

For more information about launch modes and their interaction with `Intent` flags, see Tasks and the back stack (/guide/components/tasks-and-back-stack).

`android:lockTaskMode`

Determines how the system presents this activity when the device is running in lock task mode (/work/dpc/dedicated-devices/lock-task-mode).

Android can run tasks in an immersive, kiosk-like fashion called lock task mode. When the system runs in lock task mode, device users typically can't see notifications, access non-allowlisted apps, or return to the home screen, unless the Home app is allowlisted.

Only apps that are allowlisted by a device policy controller (DPC) can run when the system is in lock task mode. System and privileged apps (https://source.android.com/devices/tech/config/perms-allowlist), however, can run in lock task mode without being allowlisted.

The value can be any one of the following `R.attr.lockTaskMode` (/reference/android/R.attr#lockTaskMode) string values:

Value	Description
`"normal"`	**Default value.** This is the default value. Tasks don't launch into lock task mode (/reference/android/app/Activity#startLockTask()).
`"never"`	Tasks don't launch into `lockTask` mode, and the device user can't pin these t
	★ **Note:** This mode is only available to system and privileged applications. Non-p
`"if_whitelisted"`	If the DPC authorizes this package using `DevicePolicyManager.setLockT` (/reference/android/app/admin/DevicePolicyManager#setLockTaskPackages(a , then this mode is identical to `always`, except that the activity needs to call `s` (/reference/android/app/Activity#stopLockTask()) before being able to finish i this package then this mode is identical to `normal`.

`"always"`	Tasks rooted at this activity always launch into lock task mode. If the system is then the new task is launched on top of the current task. Tasks launched in thi (/reference/android/app/Activity#finish()).

★ **Note:** This mode is only available to system and privileged applications. Non-p

This attribute was introduced in API level 23.

`android:maxRecents`

The maximum number of tasks rooted at this activity in the Recents screen (/guide/components/recents). When this number of entries is reached, the system removes the least-recently used instance from the Recents screen. Valid values are integers from 1 through 50, or 1 through 25 on low-memory devices. Zero is invalid. The default value is 16.

`android:maxAspectRatio`

The maximum aspect ratio the activity supports.

⚠ **Warning:** To improve the layout of apps on form factors with smallest width >= 600dp, the system ignores this attribute for apps that target Android 16 (API level 36). Your app can opt out of the Android 16 behavior, but the opt out will be eliminated in a future release. See Device compatibility mode (/guide/practices/device-compatibility-mode#android_16).

If the app runs on a device with a wider aspect ratio, the system automatically letterboxes the app, leaving portions of the screen unused so the app can run at its specified maximum aspect ratio.

Maximum aspect ratio is expressed as the decimal form of the quotient of the device's longer dimension divided by its shorter dimension. For example, if the maximum aspect ratio is 7:3, set the value of this attribute to 2.33.

On non-wearable devices, the value of this attribute needs to be 1.33 or greater. On wearable devices, it must be 1.0 or greater. Otherwise, the system ignores the

set value.

Note:

- This attribute is ignored if the activity has **`resizeableActivity`** (/reference/android/R.attr#resizeableActivity) set to true, since that means your activity supports any aspect ratio.
- Device manufacturers can override this attribute to improve the layout of apps.
- On devices with Android 16 (API level 36) or higher installed, virtual device owners (select trusted and privileged apps) can configure devices they manage to override (ignore) this attribute to improve app layout. See also Companion app streaming (https://source.android.com/docs/core/permissions/app-streaming).

See Device compatibility mode (/guide/practices/device-compatibility-mode).

For more information about this attribute, see R.attr.maxAspectRatio (/reference/android/R.attr#maxAspectRatio).

`android:minAspectRatio`

The minimum aspect ratio the activity supports.

Warning: To improve the layout of apps on form factors with smallest width >= 600dp, the system ignores this attribute for apps that target Android 16 (API level 36). Your app can opt out of the Android 16 behavior, but the opt out will be eliminated in a future release. See Device compatibility mode (/guide/practices/device-compatibility-mode#android_16).

If the app runs on a device with a narrower aspect ratio, the system automatically letterboxes the app, leaving portions of the screen unused so the app can run at its specified minimum aspect ratio.

Minimum aspect ratio is expressed as the decimal form of the quotient of the device's longer dimension divided by its shorter dimension. For example, if the display aspect ratio is 4:3, set the minimum aspect ratio value to 1.33.

The value must be greater or equal to 1.0, otherwise the system ignores the set value.

★ **Note:**

- This attribute is ignored if the activity has **`resizeableActivity`** (/reference/android/R.attr#resizeableActivity) set to true, since that means your activity supports any aspect ratio.
- Device manufacturers can override this attribute to improve the layout of apps.
- On devices with Android 16 (API level 36) or higher installed, virtual device owners (select trusted and privileged apps) can configure devices they manage to override (ignore) this attribute to improve app layout. See also Companion app streaming (https://source.android.com/docs/core/permissions/app-streaming).

See Device compatibility mode (/guide/practices/device-compatibility-mode).

For more information about this attribute, see R.attr.minAspectRatio (/reference/android/R.attr#minAspectRatio).

`android:multiprocess`

Whether an instance of the activity can be launched into the process of the component that started it. It's `"true"` if it can be, and `"false"` if not. The default value is `"false"`.

Normally, a new instance of an activity is launched into the process of the application that defined it, so all instances of the activity run in the same process. However, if this flag is set to `"true"`, instances of the activity can run in multiple processes, letting the system create instances wherever they are used, provided permissions let it—something that is almost never necessary or desirable.

`android:name`

The name of the class that implements the activity, a subclass of `Activity` (/reference/android/app/Activity). The attribute value is normally a fully qualified class name, such as, `"com.example.project.ExtracurricularActivity"`. However, as a shorthand, if the first character of the name is a period, such as

`".ExtracurricularActivity"`, it is appended to the namespace (/studio/build/configure-app-module#set-namespace) specified in the `build.gradle` file.

Once you publish your application, don't change this name (http://android-developers.blogspot.com/2011/06/things-that-cannot-change.html), unless you set `android:exported` (#exported)`="false"`. There is no default. The name must be specified.

`android:noHistory`

Whether the activity is removed from the activity stack and finished, by calling its `finish()` (/reference/android/app/Activity#finish()) method, when the user navigates away from it and it's no longer visible on screen. It's `"true"` if it is finished, and `"false"` if not. The default value is `"false"`.

A value of `"true"` means that the activity doesn't leave a historical trace. It doesn't remain in the activity stack for the task, so the user isn't able to return to it. In this case, `onActivityResult()` (/reference/android/app/Activity#onActivityResult(int, int, android.content.Intent)) is never called if you start another activity for a result from this activity.

This attribute was introduced in API level 3.

`android:parentActivityName`

The class name of the logical parent of the activity. The name here must match the class name given to the corresponding `<activity>` element's `android:name` (#nm) attribute.

The system reads this attribute to determine which activity to start when the user taps the Up button in the action bar. The system can also use this information to synthesize a back stack of activities with `TaskStackBuilder` (/reference/android/app/TaskStackBuilder).

To support API levels 4 - 16, you can also declare the parent activity with a `<meta-data>` element that specifies a value for `"android.support.PARENT_ACTIVITY"`:

```
<activity
    android:name="com.example.app.ChildActivity"
```

```
    android:label="@string/title_child_activity"
    android:parentActivityName="com.example.app.MainActivity" >
    <!-- Parent activity meta-data to support API level 4+ -->
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value="com.example.app.MainActivity" />
</activity>
```

For more information about declaring the parent activity to support Up navigation, read Providing Up Navigation (/training/implementing-navigation/ancestral).

This attribute was introduced in API level 16.

`android:persistableMode`

Defines how an instance of an activity is preserved within a containing task across device restarts.

If the root activity of a task sets this attribute's value to `persistRootOnly`, then only the root activity is preserved. Otherwise, the activities that are higher up the task's back stack (/guide/components/activities/tasks-and-back-stack) are examined; any of these activities that set this attribute's value to `persistAcrossReboots` are preserved.

If you use this attribute, you must set its value to one of the following:

Value	Description
`persistRootOnly`	**Default value.** When the system restarts, the activity task is preserved, but only the root activity's launching intent is used. When your app's launching intent loads your app's root activity, the activity doesn't receive a `PersistableBundle` (/reference/android/os/PersistableBundle) object. Therefore, don't use `onSaveInstanceState()` (/reference/android/app/Activity#onSaveInstanceState(android.os.Bundle, android.os.PersistableBundle)) to preserve the state of your app's root activity across a device restart.

★ **Note:** This attribute value affects your app's behavior only if it's set on your app's root activity.

`persistAcrossReboots` This activity's state is preserved, along with the state of each activity higher up the back stack (/guide/components/activities/tasks-and-back-stack) that has its own `persistableMode` attribute set to `persistAcrossReboots`. If an activity doesn't have a `persistableMode` attribute that is set to `persistAcrossReboots`, or if it's launched using the `Intent.FLAG_ACTIVITY_NEW_DOCUMENT` (/reference/android/content/Intent#FLAG_ACTIVITY_NEW_DOCUMENT) flag, then that activity, along with all activities higher up the back stack, aren't preserved.

When an intent loads an activity whose `persistableMode` attribute is set to `persistAcrossReboots` in your app, the activity receives a `PersistableBundle` (/reference/android/os/PersistableBundle) object in its `onCreate()` (/reference/android/app/Activity#onCreate(android.os.Bundle, android.os.PersistableBundle)) method. Therefore, you can use `onSaveInstanceState()` (/reference/android/app/Activity#onSaveInstanceState(android.os.Bundle, android.os.PersistableBundle)) to preserve the state of an activity across a device restart as long as its `persistableMode` attribute is set to `persistAcrossReboots`.

★ **Note:** This attribute value affects your app's behavior even if it's set on an activity other than your app's root activity.

`persistNever` The activity's state isn't preserved.

★ **Note:** This attribute value affects your app's behavior only if it's set on your app's root activity.

This attribute was introduced in API level 21.

`android:permission`

The name of a permission that clients must have to launch the activity or otherwise get it to respond to an intent. If a caller of `startActivity()` (/reference/android/content/Context#startActivity(android.content.Intent)) or `startActivityForResult()` (/reference/android/app/Activity#startActivityForResult(android.content.Intent, int)) isn't granted the specified permission, its intent isn't delivered to the activity.

If this attribute isn't set, the permission set by the `<application>` (/guide/topics/manifest/application-element) element's `permission` (/guide/topics/manifest/application-element#prmsn) attribute applies to the activity. If neither attribute is set, the activity isn't protected by a permission.

For more information on permissions, see the Permissions (/guide/topics/manifest/manifest-intro#perms) section of the App manifest overview and Security tips (/guide/topics/security/security).

`android:preferMinimalPostProcessing`

Specifies whether the activity prefers the connected display to perform minimal post-processing (also known as auto low latency mode or game mode). This setting is useful for applications like games or video conferencing where low latency is more important than image enhancement.

The default value is `"false"`.

For more information, see the guide on requesting and checking for low latency support (/about/versions/11/features#low_latency).

`android:process`

The name of the process in which the activity runs. Normally, all components of an application run in a default process name created for the application, and you don't need to use this attribute. But if necessary, you can override the default process name with this attribute, letting you spread your app components across multiple processes.

If the name assigned to this attribute begins with a colon (`:`), a new process, private to the application, is created when it's needed and the activity runs in that process.

If the process name begins with a lowercase character, the activity runs in a global process of that name, provided that it has permission to do so. This lets components in different applications share a process, reducing resource usage.

The `<application>` (/guide/topics/manifest/application-element) element's `process` (/guide/topics/manifest/application-element#proc) attribute can set a different default process name for all components.

`android:recreateOnConfigChanges`

Specifies the configuration changes that should trigger the system to recreate the activity. If a configuration change is specified here (and not in `android:configChanges`), the activity is recreated when that change occurs.

For more information, see the guide on recreating an activity on configuration changes (/guide/topics/resources/runtime-changes#android_17).

`android:relinquishTaskIdentity`

Whether the activity relinquishes its task identifiers to an activity above it in the task stack. A task whose root activity has this attribute set to `"true"` replaces the base `Intent` with that of the next activity in the task.

If the next activity also has this attribute set to `"true"` then it yields the base `Intent` to any activity that it launches in the same task. This continues for each activity until an activity is encountered which has this attribute set to `"false"`. The default value is `"false"`.

This attribute set to `"true"` also permits the activity's use of the `ActivityManager.TaskDescription` (/reference/android/app/ActivityManager.TaskDescription) to change labels, colors, and icons in the Recents screen (/guide/components/recents).

`android:requireContentUriPermissionFromCaller`

Specifies permissions necessary to launch this activity when passing content URIs. The default value is `none`, meaning no specific permissions are required. Setting this attribute restricts activity invocation based on the invoker's permissions. If the invoker doesn't have the required permissions, the activity start will be denied via a `SecurityException` (/reference/java/lang/SecurityException) .

Note that the enforcement works for content URIs inside `Intent.getData()` (/reference/android/content/Intent#getData()), `Intent.EXTRA_STREAM` (/reference/android/content/Intent#EXTRA_STREAM), and `Intent.getClipData()` (/reference/android/content/Intent#getClipData()).

May be a string value, using '\\;' to escape characters such as '\\n' or '\\uxxxx' for a unicode character;

Must be one of the following constant values.

Constant	Value	Description
none	0	Default, no specific permissions are required.
read	1	Enforces the invoker to have read access to the passed content URIs.
readAndWrite	4	Enforces the invoker to have both read and write access to the passed content URIs.
readOrWrite	3	Enforces the invoker to have either read or write access to the passed content URIs.
write	2	Enforces the invoker to have write access to the passed content URIs.

`android:resizeableActivity`

Specifies whether the app supports multi-window mode (/guide/topics/ui/multi-window).

Warning: To improve the layout of apps on form factors with smallest width >= 600dp, the system ignores this attribute for apps that target Android 16 (API level 36). Your app can opt out of the Android 16 behavior, but the opt out will be eliminated in a future release. See Device compatibility mode (/guide/practices/device-compatibility-mode#android_16).

You can set this attribute in either the `<activity>` or `<application>` (/guide/topics/manifest/application-element) element.

If you set this attribute to `"true"`, the user can launch the activity in split-screen and free-form modes. If you set the attribute to `"false"`, the app can't be tested or optimized for a multi-window environment. The system can still put the activity in multi-window mode with compatibility mode applied.

Setting this attribute to `"false"` doesn't guarantee that there are no other apps in multi-window mode visible on screen, such as in a picture-in-picture, or on other displays. Therefore, setting this flag doesn't mean that your app has exclusive resource access.

If your app targets API level 24 or higher and 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 app can be resized, not whether the app supports multi-window mode. 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 the minimum width and minimum height of the activity are within the multi-window requirements, the app supports multi-window mode. If `resizeableActivity="false"`, the app doesn't support multi-window mode regardless of the activity minimum width and height.

★ **Note:**

- Device manufacturers can override the API level 31 behavior to improve the layout of apps.
- On devices with Android 16 (API level 36) or higher installed, virtual device owners (select trusted and privileged apps) can configure devices they manage to override (ignore) this attribute to improve app layout. See also Companion app streaming (https://source.android.com/docs/core/permissions/app-streaming).

See Device compatibility mode (/guide/practices/device-compatibility-mode).

A task's root activity value is applied to all additional activities launched in the task. That is, if the root activity of a task is resizable, then the system treats all other activities in the task as resizable. If the root activity isn't resizable, the other activities in the task aren't resizable.

Note: A task's root activity value is applied to all additional activities launched in the task. That is, if the root activity of a task is resizable, then the system treats all other activities in the task as resizable. If the root activity isn't resizable, the other activities in the task aren't resizable.

`android:screenOrientation`

The requested orientation of the activity.

Warning: To improve the layout of apps on form factors with smallest width >= 600dp, the system ignores the following values of this attribute for apps that target Android 16 (API level 36):

- **`portrait`**
- **`landscape`**
- **`reversePortrait`**
- **`reverseLandscape`**
- **`sensorPortrait`**
- **`sensorLandscape`**
- **`userPortrait`**
- **`userLandscape`**

Your app can opt out of the Android 16 behavior, but the opt out will be eliminated in a future release. See Device compatibility mode (/guide/practices/device-compatibility-mode#android_16).

★ **Note:**

- Device manufacturers can configure devices to override (ignore) this attribute to improve the layout of apps.
- On devices with Android 16 (API level 36) or higher installed, virtual device owners (select trusted and privileged apps) can configure devices they manage to override (ignore) this attribute to improve app layout. See also Companion app streaming (https://source.android.com/docs/core/permissions/app-streaming).

See Device compatibility mode (/guide/practices/device-compatibility-mode).

When an activity fills the entire screen, the requested orientation acts as a suggestion to change the orientation on that screen to match the requested value. This can result in an orientation that differs from the screen's physical orientation in space, requiring the user to rotate the device to continue using the app. On Android 12 (API level 31) (/about/versions/12/12L/summary#dev-device-orientation-request) and higher, device manufacturers can configure individual device screens (such as the tablet-sized screen of a foldable) to ignore this suggestion, and instead force an activity to be letterboxed within the user's preferred orientation of the device. This results in the activity's orientation matching the requested one without needing the user to physically rotate their device.

In multi-window mode, the requested orientation does not act as a suggestion for the overall orientation. If the activity is letterboxed (/guide/practices/device-compatibility-mode#letterboxing), the requested orientation impacts the letterboxing applied to the activity.

The value can be any one of the following strings:

`"unspecified"`	The default value. The system chooses the orientation. The policy it uses, and therefore the choices made in specific contexts, might differ from device to device.
`"behind"`	The same orientation as the activity that's immediately beneath it in the activity stack.
`"landscape"`	Landscape orientation (the display is wider than it is tall).
`"portrait"`	Portrait orientation (the display is taller than it is wide).
`"reverseLandscape"`	Landscape orientation in the opposite direction from normal landscape. *Added in API level 9.*
`"reversePortrait"`	Portrait orientation in the opposite direction from normal portrait. *Added in API level 9.*
`"sensorLandscape"`	Landscape orientation, but can be either normal or reverse landscape based on the device sensor. The sensor is used even if the user has locked sensor-based rotation. *Added in API level 9.*
`"sensorPortrait"`	Portrait orientation, but can be either normal or reverse portrait based on the device sensor. The sensor is used even if the user has locked sensor-based rotation. However, depending on the device configuration, upside-down rotation might not be allowed. *Added in API level 9.*
`"userLandscape"`	Landscape orientation, but can be either normal or reverse landscape based on the device sensor and the user's preference. *Added in API level 18.*
`"userPortrait"`	Portrait orientation, but can be either normal or reverse portrait based on the device sensor and the user's preference. However, depending on the device configuration, upside-down rotation might not be allowed. *Added in API level 18.*
`"sensor"`	The device orientation sensor determines the orientation. The orientation of the display depends on how the user is holding the device. It changes when the user rotates the device. Some devices, though, don't rotate to all

	four possible orientations, by default. To use all four orientations, use `"fullSensor"`. The sensor is used even if the user locked sensor-based rotation.
`"fullSensor"`	The device orientation sensor determines the orientation for any of the four orientations. This is similar to `"sensor"`, except this allows for any of the four possible screen orientations regardless of what the device normally supports. For example, some devices don't normally use reverse portrait or reverse landscape, but this enables those orientations. *Added in API level 9.*
`"nosensor"`	The orientation is determined without reference to a physical orientation sensor. The sensor is ignored, so the display doesn't rotate based on how the user moves the device.
`"user"`	The user's current preferred orientation.
`"fullUser"`	If the user has locked sensor-based rotation, this behaves the same as `user`, otherwise it behaves the same as `fullSensor` and allows any of the four possible screen orientations. *Added in API level 18.*
`"locked"`	Locks the orientation to its current rotation, whatever that is. *Added in API level 18.*

★ **Note:** When you declare one of the landscape or portrait values, it is considered a hard requirement for the orientation in which the activity runs. The value you declare enables filtering by services such as Google Play, so your application is available only to devices that support the orientation required by your activities. For example, if you declare either `"landscape"`, `"reverseLandscape"`, or `"sensorLandscape"`, then your application is available only to devices that support landscape orientation.

Also explicitly declare that your application requires either portrait or landscape orientation with the `<uses-feature>` (/guide/topics/manifest/uses-feature-element) Element, such as `<uses-feature android:name="android.hardware.screen.portrait"/>`. This is a filtering behavior provided by Google Play and other services that support it, and the

platform itself doesn't control whether your app can install when a device supports only certain orientations.

`android:showForAllUsers`

Whether the activity is shown when the device's current user is different than the user who launched the activity. You can set this attribute to a literal value, like `"true"` or `"false"`, or you can set the attribute to a resource or theme attribute that contains a boolean value.

This attribute was added in API level 23.

`android:stateNotNeeded`

Whether the activity can be terminated and successfully restarted without having saved its state. It's `"true"` if it can be restarted without reference to its previous state, and `"false"` if its previous state is required. The default value is `"false"`.

Normally, before an activity is temporarily shut down to save resources, its `onSaveInstanceState()` (/reference/android/app/Activity#onSaveInstanceState(android.os.Bundle)) method is called. This method stores the current state of the activity in a `Bundle` (/reference/android/os/Bundle) object, which is then passed to `onCreate()` (/reference/android/app/Activity#onCreate(android.os.Bundle)) when the activity is restarted. If this attribute is set to `"true"`, `onSaveInstanceState()` might not be called, and `onCreate()` is passed `null` instead of the `Bundle`, as it is when the activity starts for the first time.

A `"true"` setting means that the activity can be restarted without retained state. For example, the activity that displays the Home screen uses this setting to make sure that it doesn't get removed if it crashes for some reason.

`android:supportsPictureInPicture`

Specifies whether the activity supports picture-in-picture (/training/tv/playback/picture-in-picture) display.

`android:taskAffinity`

The task that the activity has an affinity for. Activities with the same affinity conceptually belong to the same task, to the same "application" from the user's

perspective. The affinity of a task is determined by the affinity of its root activity.

The affinity determines two things: the task that the activity is re-parented to (see the `allowTaskReparenting` (/guide/topics/manifest/activity-element#reparent) attribute) and the task that houses the activity when it launches with the `FLAG_ACTIVITY_NEW_TASK` (/reference/android/content/Intent#FLAG_ACTIVITY_NEW_TASK) flag.

By default, all activities in an application have the same affinity. You can set this attribute to group them differently, and even place activities defined in different applications within the same task. To specify that the activity doesn't have an affinity for any task, set it to an empty string.

If this attribute isn't set, the activity inherits the affinity set for the application. See the `<application>` (/guide/topics/manifest/application-element) element's `taskAffinity` (/guide/topics/manifest/application-element#aff) attribute. The name of the default affinity for an application is the namespace (/studio/build/configure-app-module#set-namespace) set in the `build.gradle` file.

`android:theme`

A reference to a style resource defining an overall theme for the activity. This automatically sets the activity's context to use this `theme` (/reference/android/content/Context#setTheme(int)) and might also cause "starting" animations prior to the activity being launched, to better match what the activity actually looks like.

If this attribute isn't set, the activity inherits the theme set for the application as a whole, from the `<application>` (/guide/topics/manifest/application-element) element's `theme` (/guide/topics/manifest/application-element#theme) attribute. If that attribute is also not set, the default system theme is used. For more information, see Styles and themes (/guide/topics/ui/themes).

`android:uiOptions`

Extra options for an activity's UI. Must be one of the following values.

Value	Description

`"none"`	No extra UI options. This is the default.
`"splitActionBarWhenNarrow"`	Adds a bar at the bottom of the screen to display action items in the *app bar*, also known as the *action bar*, when constrained for horizontal space, such as when in portrait mode on a handset. Instead of a small number of action items appearing in the app bar at the top of the screen, the app bar is split into the top navigation section and the bottom bar for action items. This means a reasonable amount of space is made available not only for the action items, but also for navigation and title elements at the top. Menu items are not split across the two bars. They always appear together.

For more information about the app bar, see Add the app bar (/training/appbar).

This attribute was added in API level 14.

`android:windowSoftInputMode`

How the main window of the activity interacts with the window containing the on-screen soft keyboard. The setting for this attribute affects two things:

- Whether the soft keyboard is hidden or visible when the activity becomes the focus of user attention.
- Whether the activity's main window is resized smaller to make room for the soft keyboard or its contents pan to make the current focus visible when part of the window is covered by the soft keyboard.

The setting must be one of the values listed in the following table or a combination of one `"state..."` value plus one `"adjust..."` value. Setting multiple values in either group, such as multiple `"state..."` values, has undefined results. Individual values are separated by a vertical bar (`|`), as shown in the following example:

```
<activity android:windowSoftInputMode="stateVisible|adjustResize" ... >
```

Values set here (other than `"stateUnspecified"` and `"adjustUnspecified"`) override values set in the theme.

Value	Description
`"stateUnspecified"`	Whether the soft keyboard is hidden or visible isn't specified. The system chooses an appropriate state or relies on the setting in the theme. This is the default setting for the behavior of the soft keyboard.
`"stateUnchanged"`	The soft keyboard is kept in whatever state it was last in, visible or hidden, when the activity comes to the fore.
`"stateHidden"`	The soft keyboard is hidden when the user chooses the activity—that is, when the user affirmatively navigates forward to the activity, rather than backing into it when leaving another activity.
`"stateAlwaysHidden"`	The soft keyboard is always hidden when the activity's main window has input focus.
`"stateVisible"`	The soft keyboard is made visible when the user chooses the activity—that is, when the user affirmatively navigates forward to the activity, rather than backing into it when leaving another activity.
`"stateAlwaysVisible"`	The soft keyboard is visible when the window receives input focus.
`"adjustUnspecified"`	Whether the activity's main window resizes to make room for the soft keyboard or the contents of the window pan to make the current focus visible on-screen is unspecified. The system automatically selects one of these modes depending on whether the content of the window has any layout views that can scroll their contents. If there is such a view, the window resizes, on the assumption that scrolling can make all of the window's contents visible within a smaller area. This is the default setting for the behavior of the main window.
`"adjustResize"`	The activity's main window is always resized to make room for the soft keyboard on screen.
`"adjustPan"`	The activity's main window isn't resized to make room for the soft keyboard. Rather, the contents of the window automatically pan so that the current focus is never obscured by the keyboard, and users can always see what they are typing. This is generally less desirable than resizing, because the

	user might need to close the soft keyboard to get at and interact with obscured parts of the window.
`"adjustNothing"`	The activity's main window isn't resized or panned to make room for the soft keyboard. The activity is responsible for making room for the soft keyboard using the window insets. For activities that handle window insets correctly this gives the most control over how the window's contents are displayed on the screen.

This attribute was introduced in API level 3.

introduced in:

API level 1 for all attributes except `noHistory` (#nohist) and `windowSoftInputMode` (#wsoft), which were added in API level 3.

see also:

`<application>` (/guide/topics/manifest/application-element)
`<activity-alias>` (/guide/topics/manifest/activity-alias-element)