<activity>

SYNTAX:

```
<activity android:allowTaskReparenting=["true" | "false"]
          android:alwaysRetainTaskState=["true" | "false"]
          android:clearTaskOnLaunch=["true" | "false"]
          android:configChanges=["mcc", "mnc", "locale",
                                 "touchscreen", "keyboard", "keyboardHidd
                                 "navigation", "screenLayout", "fontScale
                                 "orientation", "screenSize", "smallestSc
          android:enabled=["true" | "false"]
          android:excludeFromRecents=["true" | "false"]
          android:exported=["true" | "false"]
          android:finishOnTaskLaunch=["true" | "false"]
          android:hardwareAccelerated=["true" | "false"]
          android:icon="drawable resource"
          android:label="string resource"
          android:launchMode=["multiple" | "singleTop" |
                              "singleTask" | "singleInstance"]
          android:multiprocess=["true" | "false"]
          android:name="string"
          android:noHistory=["true" | "false"]
          android:parentActivityName="string"
          android:permission="string"
          android:process="string"
          android:screenOrientation=["unspecified" | "behind" |
                                     "landscape" | "portrait" |
                                     "reverseLandscape" | "reversePortrai
                                     "sensorLandscape" | "sensorPortrait"
                                     "userLandscape" | "userPortrait" |
                                     "sensor" | "fullSensor" | "nosensor"
                                     "user" | "fullUser" | "locked"]
          android:stateNotNeeded=["true" | "false"]
          android:taskAffinity="string"
          android:theme="resource or theme"
          android:uiOptions=["none" | "splitActionBarWhenNarrow"]
          android:windowSoftInputMode=["stateUnspecified",
                                       "stateUnchanged", "stateHidden",
                                       "stateAlwaysHidden", "stateVisible
                                       "stateAlwaysVisible", "adjustUnspe
                                       "adjustResize", "adjustPan"] >
    . . .
</activity>
```

CONTAINED IN:

<application>

CAN CONTAIN:

<intent-filter>
<meta-data>

DESCRIPTION:

Declares an activity (an 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 are not declared there will not be seen by the system and will never be run.

ATTRIBUTES:

`android:allowTaskReparenting`

Whether or not 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 — "`true`" if it can move, and "`false`" if it must remain with the task where it started.

If this attribute is not set, the value set by the corresponding `allowTaskReparenting` (/guide/topics/manifest/application-element.html#reparent) attribute of the `<application>` (/guide/topics/manifest/application-element.html) 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, it's used to cause the activities of an application to move to the main task associated with that application.

For example, if an e-mail 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 e-mail task. If it's reparented to the browser task, it will be shown when the browser next comes to the front, and will be absent when the e-mail 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 or not the state of the task that the activity is in will always be maintained by the system — "`true`" if it will be, and "`false`" if the system is allowed to 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 (removes 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 will always return to the task in its last state, regardless of how they get there. This is useful, for example, in an application like the web browser where there is a lot of state (such as multiple open tabs) that users would not like to lose.

`android:clearTaskOnLaunch`

Whether or not all activities will be removed from the task, except for the root activity, whenever 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 again, 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 may be cleared of activities in some situations (see the `alwaysRetainTaskState` (#always) attribute), but

not always.

Suppose, for example, that someone launches activity P from the home screen, and from there goes to activity Q. The user next presses *Home*, and then returns to activity P. Normally, the user would see 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 (Q in this case) were removed when the user pressed *Home* and the task went to the background. 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, as described above.

`android:configChanges`

Lists configuration changes that the activity will handle itself. When a configuration change occurs at runtime, the activity is shut down and restarted by default, but declaring a configuration with this attribute will prevent the activity from being restarted. Instead, the activity remains running and its `onConfigurationChanged()` method is called.

> **Note:** Using this attribute should be avoided and used only as a last resort. Please read Handling Runtime Changes (/guide/topics/resources/runtime-changes.html) for more information about how to properly handle a restart due to a configuration change.

Any or all of the following strings are valid values for this attribute. Multiple values are separated by '|' — for example, "`locale|navigation|orientation`".

Value	Description
"`mcc`"	The IMSI mobile country code (MCC) has changed — a SIM has been detected and updated the MCC.
"`mnc`"	The IMSI mobile network code (MNC) has changed — a SIM has been detected and updated the MNC.
"`locale`"	The locale has changed — the user has selected a new language that text should be displayed in.
"`touchscreen`"	The touchscreen has changed. (This should never normally happen.)
"`keyboard`"	The keyboard type has changed — for example, the user has plugged in an external keyboard.
"`keyboardHidden`"	The keyboard accessibility has changed — for example, the user has revealed the hardware keyboard.
"`navigation`"	The navigation type (trackball/dpad) has changed. (This should never normally happen.)
"`screenLayout`"	The screen layout has changed — this might be caused by a different display being activated.
"`fontScale`"	The font scaling factor has changed — the user has selected a new global font size.
"`uiMode`"	The user interface mode has changed — this can be caused when the user places the device into a desk/car dock or when the night mode changes. See `UiModeManager`. *Added in API level 8*.
"`orientation`"	The screen orientation has changed — the user has rotated the device. **Note:** If your application targets API level 13 or higher (as declared by the `minSdkVersion` (/guide/topics/manifest/uses-sdk-element.html#min) and `targetSdkVersion` (/guide/topics/manifest

> /uses-sdk-element.html#target) attributes), then you should also declare the "screenSize" configuration, because it also changes when a device switches between portrait and landscape orientations.

"screenSize"	The current available screen size has changed. This represents a change in the currently available size, relative to the current aspect ratio, so will change when the user switches between landscape and portrait. However, if your application targets API level 12 or lower, then your activity always handles this configuration change itself (this configuration change does not restart your activity, even when running on an Android 3.2 or higher device). *Added in API level 13.*
"smallestScreenSize"	The physical screen size has changed. This represents a change in size regardless of orientation, so will only change when the actual physical screen size has changed such as switching to an external display. A change to this configuration corresponds to a change in the smallestWidth configuration. However, if your application targets API level 12 or lower, then your activity always handles this configuration change itself (this configuration change does not restart your activity, even when running on an Android 3.2 or higher device). *Added in API level 13.*
"layoutDirection"	The layout direction has changed. For example, changing from left-to-right (LTR) to right-to-left (RTL). *Added in API level 17.*

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

android:enabled
: Whether or not the activity can be instantiated by the system — "true" if it can be, and "false" if not. The default value is "true".

  The <application> (/guide/topics/manifest/application-element.html) element has its own enabled (/guide/topics/manifest/application-element.html#enabled) attribute that applies to all application components, including activities. The <application> (/guide/topics/manifest/application-element.html) 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 cannot be instantiated.

android:excludeFromRecents
: Whether or not the task initiated by this activity should be excluded from the list of recently used applications ("recent apps"). That is, when this activity is the root activity of a new task, this attribute determines whether the task should not appear in the list of recent apps. Set "true" if the task should be *excluded* from the list; set "false" if it should be *included*. The default value is "false".

android:exported

Whether or not the activity can be launched by components of other applications — "`true`" if it can be, and "`false`" if not. If "`false`", the activity can be launched only by components of the same application or applications with the same user ID.

The default value depends on whether the activity contains intent filters. The absence of any filters means that the activity can be invoked only by specifying its exact class name. This implies that the activity is intended only for application-internal use (since others would not know the class name). So in this case, the default value is "`false`". On the other hand, the presence of at least one filter implies that the activity is intended for external use, so the default value is "`true`".

This attribute is not the only way to limit an activity's exposure to other applications. You can also use a permission to limit the external entities that can invoke the activity (see the `permission` (/guide/topics/manifest/activity-element.html#prmsn) attribute).

`android:finishOnTaskLaunch`

Whether or not an existing instance of the activity should be shut down (finished) whenever the user again launches its task (chooses the task on the home screen) — "`true`" if it should be shut down, and "`false`" if not. The default value is "`false`".

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

`android:hardwareAccelerated`

Whether or not hardware-accelerated rendering should be enabled for this Activity — "`true`" if it should be enabled, and "`false`" if not. The default value is "`false`".

Starting from Android 3.0, 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 do not explicitly make use the framework's OpenGL libraries. Because of the increased resources required to enable hardware acceleration, your app will consume more RAM.

Note that not all of the OpenGL 2D operations are accelerated. If you enable the hardware-accelerated renderer, test your application to ensure that it can make use of 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 (see the `android:label` attribute).

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

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

`android:label`

A user-readable label for the activity. The label is displayed on-screen when the activity must be represented to the user. It's often displayed along with the activity icon.

If this attribute is not set, the label set for the application as a whole is used instead (see the `<application>` (/guide/topics/manifest/application-element.html) element's `label` (/guide/topics/manifest/application-element.html#label) attribute).

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

The label should be 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 on how the activity should be launched. There are four modes that work in conjunction with activity flags (`FLAG_ACTIVITY_*` constants) in `Intent` objects to determine what should happen when the activity is called upon to handle an intent. They are:

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

The default mode is "`standard`".

As shown in the table below, the modes fall into two main groups, with "`standard`" and "`singleTop`" activities on one side, and "`singleTask`" and "`singleInstance`" 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 stack. Typically, they're launched into the task that called `startActivity()` (/reference/android/content/Context.html#startActivity(android.content.Intent)) (unless the Intent object contains a `FLAG_ACTIVITY_NEW_TASK` (/reference/android/content/Intent.html#FLAG_ACTIVITY_NEW_TASK) instruction, in which case a different task is chosen — see the taskAffinity (#aff) attribute).

In contrast, "`singleTask`" and "`singleInstance`" activities can only begin a task. They are always at the root of the activity stack. Moreover, the device can hold only one instance of the activity at a time — only one such task.

The "`standard`" and "`singleTop`" modes differ from each other in just 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 may 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 will receive the new intent (in an `onNewIntent()` (/reference/android/app/Activity.html#onNewIntent(android.content.Intent)) call); a new instance is not created. In other circumstances — for example, 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 would be created and pushed on the stack.

The "`singleTask`" and "`singleInstance`" modes also differ from each other in only one

respect: A "`singleTask`" activity allows other activities to be part of its task. It's always at the root of its task, but other activities (necessarily "`standard`" and "`singleTop`" activities) can be launched into that task. A "`singleInstance`" activity, on the other hand, permits no other activities to be part of its task. It's 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` was 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 of the activity in the target task and routes the intent to it.
	"`singleTop`"	Conditionally	If an instance of the activity already exists at the top of the target task, the system routes the intent to that instance through a call to its `onNewIntent()` method, rather than creating a new instance of the activity.
Specialized launches *(not recommended for general use)*	"`singleTask`"	No	The system creates the activity at the root of a new task and routes the intent to it. However, if an instance of the activity already exists, the system routes the intent to existing instance through a call to its `onNewIntent()` method, rather than creating a new one.
	"`singleInstance`"	No	Same as "`singleTask`", except that the system doesn't launch any other activities into the task holding the instance. The activity is always the single and only member of its task.

As shown in the table above, `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` and `singleInstance` — are not appropriate for most applications, since 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 on launch modes and their interaction with Intent flags, see the Tasks and Back Stack (/guide/components/tasks-and-back-stack.html) document.

`android:multiprocess`

Whether an instance of the activity can be launched into the process of the component that started it — "`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, allowing the system to create instances wherever they are used (provided permissions allow it), something that is almost never necessary or desirable.

`android:name`

The name of the class that implements the activity, a subclass of `Activity`. The attribute value should be 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 (for example, "`.ExtracurricularActivity`"), it is appended to the package name specified in the `<manifest>` element.

Once you publish your application, you should not change this name (http://android-developers.blogspot.com/2011/06/things-that-cannot-change.html) (unless you've set `android:exported` (#exported)="`false`").

There is no default. The name must be specified.

`android:noHistory`

Whether or not the activity should be removed from the activity stack and finished (its `finish()` method called) when the user navigates away from it and it's no longer visible on screen — "`true`" if it should be finished, and "`false`" if not. The default value is "`false`".

A value of "`true`" means that the activity will not leave a historical trace. It will not remain in the activity stack for the task, so the user will not be able to return to it.

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` attribute.

The system reads this attribute to determine which activity should be started when the use presses 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.html).

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`". For example:

```
<activity
    android:name="com.example.app.ChildActivity"
    android:label="@string/title_child_activity"
    android:parentActivityName="com.example.myfirstapp.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.html).

This attribute was introduced in API Level 16.

`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()` or `startActivityForResult()` has not been granted the specified permission, its intent will not be delivered to the activity.

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

For more information on permissions, see the Permissions (/guide/topics/manifest/manifest-intro.html#sectperm) section in the introduction and another document, Security and Permissions (/guide/topics/security/security.html).

android:process
The name of the process in which the activity should run. Normally, all components of an application run in a default process name created for the application and you do not need to use this attribute. But if necessary, you can override the default process name with this attribute, allowing you to 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 will run in a global process of that name, provided that it has permission to do so. This allows components in different applications to share a process, reducing resource usage.

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

android:screenOrientation
The orientation of the activity's display on the device.

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, may 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. *Added in API level 9.*
"sensorPortrait"	Portrait orientation, but can be either normal or reverse portrait based on the device sensor. *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 sensor preference. If the user has locked sensor-based rotation, this behaves the same as landscape, otherwise it behaves the same as sensorLandscape. *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 sensor preference. If the user has locked sensor-based rotation, this behaves the same as portrait,

	otherwise it behaves the same as `sensorPortrait`. *Added in API level 18.*
`"sensor"`	The orientation is determined by the device orientation sensor. 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, will not rotate to all four possible orientations, by default. To allow all four orientations, use `"fullSensor"`.
`"fullSensor"`	The orientation is determined by the device orientation sensor for any of the 4 orientations. This is similar to `"sensor"` except this allows any of the 4 possible screen orientations, regardless of what the device will normally do (for example, some devices won't normally use reverse portrait or reverse landscape, but this enables those). *Added in API level 9.*
`"nosensor"`	The orientation is determined without reference to a physical orientation sensor. The sensor is ignored, so the display will not rotate based on how the user moves the device. Except for this distinction, the system chooses the orientation using the same policy as for the "`unspecified`" setting.
`"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 4 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. As such, 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 will be available only to devices that support landscape orientation. However, you should also explicitly declare that your application requires either portrait or landscape orientation with the `<uses-feature>` (/guide/topics/manifest/uses-feature-element.html) element. For example, `<uses-feature android:name="android.hardware.screen.portrait"/>`. This is purely a filtering behavior provided by Google Play (and other services that support it) and the platform itself does not control whether your app can be installed when a device supports only certain orientations.

`android:stateNotNeeded`

Whether or not the activity can be killed and successfully restarted without having saved its state — "`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.html#onSaveInstanceState(android.os.Bundle)) method is called. This method stores the current state of the activity in a `Bundle` (/reference/android/os/Bundle.html) object, which is then passed to `onCreate()` (/reference/android/app/Activity.html#onCreate(android.os.Bundle)) when the activity is restarted. If this attribute is set to "`true`", `onSaveInstanceState()` may not be called and `onCreate()` will be passed `null` instead of the Bundle — just as it was when the activity started for the first time.

A "true" setting ensures that the activity can be restarted in the absence of retained state. For example, the activity that displays the home screen uses this setting to make sure that it does not get removed if it crashes for some reason.

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.html#reparent) attribute) and the task that will house the activity when it is launched with the FLAG_ACTIVITY_NEW_TASK (/reference/android/content/Intent.html#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 does not have an affinity for any task, set it to an empty string.

If this attribute is not set, the activity inherits the affinity set for the application (see the <application> (/guide/topics/manifest/application-element.html) element's taskAffinity (/guide/topics/manifest/application-element.html#aff) attribute). The name of the default affinity for an application is the package name set by the <manifest> (/guide/topics/manifest/manifest-element.html) element.

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 (see setTheme(), and may also cause "starting" animations prior to the activity being launched (to better match what the activity actually looks like).

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

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"	Add a bar at the bottom of the screen to display action items in the ActionBar, 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 action bar at the top of the screen, the action bar is split into the top navigation section and the bottom bar for action items. This ensures 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 action bar, see the Action Bar (/guide/topics/ui/actionbar.html) developer guide.

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:

- The state of the soft keyboard — whether it is hidden or visible — when the activity becomes the focus of user attention.
- The adjustment made to the activity's main window — whether it is resized smaller to make room for the soft keyboard or whether 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 — multiple "`state...`" values, for example — has undefined results. Individual values are separated by a vertical bar (|). For example:

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

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

Value	Description
"`stateUnspecified`"	The state of the soft keyboard (whether it is hidden or visible) is not specified. The system will choose an appropriate state or rely 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, whether 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 backs into it because of leaving another activity.
"`stateAlwaysHidden`"	The soft keyboard is always hidden when the activity's main window has input focus.
"`stateVisible`"	The soft keyboard is visible when that's normally appropriate (when the user is navigating forward to the activity's main window).
"`stateAlwaysVisible`"	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 backs into it because of leaving another activity.
"`adjustUnspecified`"	It is unspecified whether the activity's main window resizes to make room for the soft keyboard, or whether the contents of the window pan to make the current focus visible on-screen. The system will automatically select 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 will be resized, 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 is not resized to make room for the soft keyboard. Rather, the contents of the window are automatically panned 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 may need to close the soft keyboard to get at and interact with obscured parts of the window.

This attribute was introduced in API Level 3.

INTRODUCED IN:

API Level 1 for all attributes except for noHistory and windowSoftInputMode, which were added in API Level 3.

SEE ALSO:

<application>
<activity-alias>