<supports-screens>

SYNTAX:

```
<supports-screens android:resizeable=["true"| "false"]
                  android:smallScreens=["true" | "false"]
                  android:normalScreens=["true" | "false"]
                  android:largeScreens=["true" | "false"]
                  android:xlargeScreens=["true" | "false"]
                  android:anyDensity=["true" | "false"]
                  android:requiresSmallestWidthDp="integer"
                  android:compatibleWidthLimitDp="integer"
                  android:largestWidthLimitDp="integer"/>
```

CONTAINED IN:

`<manifest>`

DESCRIPTION:

Lets you specify the screen sizes your application supports and enable screen compatibility mode for screens larger than what your application supports. It's important that you always use this element in your application to specify the screen sizes your application supports.

An application "supports" a given screen size if it resizes properly to fill the entire screen. Normal resizing applied by the system works well for most applications and you don't have to do any extra work to make your application work on screens larger than a handset device. However, it's often important that you optimize your application's UI for different screen sizes by providing alternative layout resources. For instance, you might want to modify the layout of an activity when it is on a tablet compared to when running on a handset device.

However, if your application does not work well when resized to fit different screen sizes, you can use the attributes of the `<supports-screens>` element to control whether your application should be distributed to smaller screens or have its UI scaled up ("zoomed") to fit larger screens using the system's screen compatibility mode. When you have not designed for larger screen sizes and the normal resizing does not achieve the appropriate results, screen compatibility mode will scale your UI by emulating a *normal* size screen and medium density, then zooming in so that it fills the entire screen. Beware that this causes pixelation and blurring of your UI, so it's better if you optimize your UI for large screens.

Note: Android 3.2 introduces new attributes: `android:requiresSmallestWidthDp`, `android:compatibleWidthLimitDp`, and `android:largestWidthLimitDp`. If you're developing your application for Android 3.2 and higher, you should use these attributes to declare your screen size support, instead of the attributes based on generalized screen sizes.

For more information about how to properly support different screen sizes so that you can avoid using screen compatibility mode with your application, read Supporting Multiple Screens.

ATTRIBUTES:

`android:resizeable`

Indicates whether the application is resizeable for different screen sizes. This attribute is true, by default. If set false, the system will run your application in screen compatibility mode on large screens.

This attribute is deprecated. It was introduced to help applications transition from Android 1.5 to 1.6, when support for multiple screens was first introduced. You should not use it.

`android:smallScreens`

Indicates whether the application supports smaller screen form-factors. A small screen is defined as one with a smaller aspect ratio than the "normal" (traditional HVGA) screen. An application that does not support small screens *will not be available* for small screen devices from external services (such as Google Play), because there is little the platform can do to make such an application work on a smaller screen. This is `"true"` by default.

`android:normalScreens`

Indicates whether an application supports the "normal" screen form-factors. Traditionally this is an HVGA medium density screen, but WQVGA low density and WVGA high density are also considered to be normal. This attribute is "true" by default.

`android:largeScreens`

Indicates whether the application supports larger screen form-factors. A large screen is defined as a screen that is significantly larger than a "normal" handset screen, and thus might require some special care on the application's part to make good use of it, though it may rely on resizing by the system to fill the screen.

The default value for this actually varies between some versions, so it's better if you explicitly declare this attribute at all times. Beware that setting it "false" will generally enable screen compatibility mode.

`android:xlargeScreens`

Indicates whether the application supports extra large screen form-factors. An xlarge screen is defined as a screen that is significantly larger than a "large" screen, such as a tablet (or something larger) and may require special care on the application's part to make good use of it, though it may rely on resizing by the system to fill the screen.

The default value for this actually varies between some versions, so it's better if you explicitly declare this attribute at all times. Beware that setting it "false" will generally enable screen compatibility mode.

This attribute was introduced in API level 9.

`android:anyDensity`

Indicates whether the application includes resources to accommodate any screen density.

For applications that support Android 1.6 (API level 4) and higher, this is "true" by default and **you should not set it "false"** unless you're absolutely certain that it's necessary for your application to work. The only time it might be necessary to disable this is if your app directly manipulates bitmaps (see the Supporting Multiple Screens document for more information).

`android:requiresSmallestWidthDp`

Specifies the minimum smallestWidth required. The smallestWidth is the shortest dimension of the screen space (in `dp` units) that must be available to your application UI—that is, the shortest of the available screen's two dimensions. So, in order for a device to be considered compatible with your application, the device's smallestWidth must be equal to or greater than this value. (Usually, the value you supply for this is the "smallest width" that your layout supports, regardless of the screen's current orientation.)

For example, a typical handset screen has a smallestWidth of 320dp, a 7" tablet has a smallestWidth of 600dp, and a 10" tablet has a smallestWidth of 720dp. These values are generally the smallestWidth because they are the shortest dimension of the screen's available space.

The size against which your value is compared takes into account screen decorations and system UI. For example, if the device has some persistent UI elements on the display, the system declares the device's smallestWidth as one that is smaller than the actual screen size, accounting for these UI elements because those are screen pixels not available for your UI. Thus, the value you use should be the minimum width required by your layout, regardless of the screen's current orientation.

If your application properly resizes for smaller screen sizes (down to the *small* size or a minimum width of 320dp), you do not need to use this attribute. Otherwise, you should use a value for this attribute that matches the smallest value used by your application for the smallest screen width qualifier (`sw<N>dp`).

> **Caution:** The Android system does not pay attention to this attribute, so it does not affect how your application behaves at runtime. Instead, it is used to enable filtering for your application on services such as Google Play. However, **Google Play currently does not support this attribute for filtering** (on Android 3.2), so you should continue using the other size attributes if your application does not support small screens.

This attribute was introduced in API level 13.

`android:compatibleWidthLimitDp`

This attribute allows you to enable screen compatibility mode as a user-optional feature by specifying the maximum "smallest screen width" for which your application is designed. If the smallest side of a device's available screen is greater than your value here, users can still install your application, but are offered to run it in screen compatibility mode. By default, screen compatibility mode is disabled and your layout is resized to fit the screen as usual, but a button is available in the system bar that allows the user to toggle screen compatibility mode on and off.

If your application is compatible with all screen sizes and its layout properly resizes, you do not need to use this attribute.

> **Note:** Currently, screen compatibility mode emulates only handset screens with a 320dp width, so screen compatibility mode is not applied if your value for `android:compatibleWidthLimitDp` is larger than 320.

This attribute was introduced in API level 13.

`android:largestWidthLimitDp`

This attribute allows you to force-enable screen compatibility mode by specifying the maximum "smallest screen width" for which your application is designed. If the smallest side of a device's available screen is greater than your value here, the application runs in screen compatibility mode with no way for the user to disable it.

If your application is compatible with all screen sizes and its layout properly resizes, you do not need to use this attribute. Otherwise, you should first consider using the `android:compatibleWidthLimitDp` attribute. You should use the `android:largestWidthLimitDp` attribute only when your application is functionally broken when resized for larger screens and screen compatibility mode is the only way that users should use your application.

> **Note:** Currently, screen compatibility mode emulates only handset screens with a 320dp width, so screen compatibility mode is not applied if your value for `android:largestWidthLimitDp` is larger than 320.

This attribute was introduced in API level 13.

INTRODUCED IN:

API Level 4

SEE ALSO:

- Supporting Multiple Screens
- `DisplayMetrics`