# Migrating to WebView in Android 4.4

In this document



Android 4.4 (API level 19) introduces a new version of `WebView` that is based on Chromium. This change upgrades `WebView` performance and standards support for HTML5, CSS3, and JavaScript to match the latest web browsers. Any apps using `WebView` will inherit these upgrades when running on Android 4.4 and higher.

This document describes additional changes to `WebView` that you should be aware of if you set your `targetSdkVersion` to "19" or higher.

> **Note:** If your `targetSdkVersion` is set to "18" or lower, `WebView` operates in "quirks mode" in order to avoid some of the behavior changes described below, as closely as possible—while still providing your app the performance and web standards upgrades. Beware, though, that single and narrow column layouts and default zoom levels are **not supported at all** on Android 4.4, and there may be other behavioral differences that have not been identified, so be sure to test your app on Android 4.4 or higher even if you keep your `targetSdkVersion` set to "18" or lower.

To help you work through any issues you may encounter when migrating your app to `WebView` in Android 4.4, you can enable remote debugging through Chrome on your desktop by calling `setWebContentsDebuggingEnabled()`. This new feature in `WebView` allows you to inspect and analyze your web content, scripts, and network activity while running in a `WebView`. For more information, see Remote Debugging on Android.

## User Agent Changes

If you serve content to your `WebView` based on the user agent, you should to be aware of the user agent string has changed slightly and now includes the Chrome version:

```
Mozilla/5.0 (Linux; Android 4.4; Nexus 4 Build/KRT16H) AppleWebKit/537.36
(KHTML, like Gecko) Version/4.0 Chrome/30.0.0.0 Mobile Safari/537.36
```

If you need to retrieve the user agent but don't need to store it for your app or do not want to instantiate `WebView`, you should use the static method, `getDefaultUserAgent()`. However, if you intend to override the user agent string in your `WebView`, you may instead want to use `getUserAgentString()`.

## Multi-threading and Thread Blocking

If you call methods on `WebView` from any thread other than your app's UI thread, it can cause unexpected results. For example, if your app uses multiple threads, you can use the `runOnUiThread()` method to ensure your code executes on the UI thread:

```
runOnUiThread(new Runnable() {
    @Override
    public void run() {
        // Code for WebView goes here
    }
});
```

Also be sure that you never block the UI thread. A situation in which some apps make this mistake is while waiting for a JavaScript callback. For example, **do not** use code like this:

```
// This code is BAD and will block the UI thread
webView.loadUrl("javascript:fn()");
while(result == null) {
  Thread.sleep(100);
}
```

You can instead use a new method, `evaluateJavascript()`, to run JavaScript asynchronously.

# Custom URL Handling

The new `WebView` applies additional restrictions when requesting resources and resolving links that use a custom URL scheme. For example, if you implement callbacks such as `shouldOverrideUrlLoading()` or `shouldInterceptRequest()`, then `WebView` invokes them only for valid URLs.

If you are using a custom URL scheme or a base URL and notice that your app is receiving fewer calls to these callbacks or failing to load resources on Android 4.4, ensure that the requests specify valid URLs that conform to RFC 3986.

For example, the new `WebView` may not call your `shouldOverrideUrlLoading()` method for links like this:

```
<a href="showProfile">Show Profile</a>
```

The result of the user clicking such a link can vary:

- If you loaded the page by calling `loadData()` or `loadDataWithBaseURL()` with an invalid or null base URL, then you will not receive the `shouldOverrideUrlLoading()` callback for this type of link on the page.

  > **Note:** When you use `loadDataWithBaseURL()` and the base URL is invalid or set null, all links in the content you are loading must be absolute.

- If you loaded the page by calling `loadUrl()` or provided a valid base URL with `loadDataWithBaseURL()`, then you will receive the `shouldOverrideUrlLoading()` callback for this type of link on the page, but the URL you receive will be absolute, relative to the current page. For example, the URL you receive will be `"http://www.example.com/showProfile"` instead of just `"showProfile"`.

Instead of using a simple string in a link as shown above, you can use a custom scheme such as the following:

```
<a href="example-app:showProfile">Show Profile</a>
```

You can then handle this URL in your `shouldOverrideUrlLoading()` method like this:

```
// The URL scheme should be non-hierarchical (no trailing slashes)
private static final String APP_SCHEME = "example-app:";

@Override
public boolean shouldOverrideUrlLoading(WebView view, String url) {
    if (url.startsWith(APP_SCHEME)) {
        urlData = URLDecoder.decode(url.substring(APP_SCHEME.length()), "UTF-8");
        respondToData(urlData);
        return true;
    }
    return false;
}
```

If you can't alter the HTML then you may be able to use `loadDataWithBaseURL()` and set a base URL consisting of a custom scheme and a valid host, such as `"example-app://<valid_host_name>/"`. For example:

```
webView.loadDataWithBaseURL("example-app://example.co.uk/", HTML_DATA,
        null, "UTF-8", null);
```

The valid host name should conform to RFC 3986 and it's important to include the trailing slash at the end, otherwise, any requests from the loaded page may be dropped.

# Viewport Changes

## Viewport target-densitydpi no longer supported

Previously, `WebView` supported a viewport property called `target-densitydpi` to help web pages specify their intended screen density. This property is no longer supported and you should migrate to using standard solutions with images and CSS as discussed in Pixel-Perfect UI in the WebView.

## Viewport zooms in when small

Previously, if you set your viewport width to a value less than or equal to "320" it would be set to "device-width", and if you set the viewport height to a value less than or equal to the `WebView` height, it would be set to "device-height". However, when running in the new `WebView`, the width or height value is adhered and the `WebView` zooms in to fill the screen width.

## Multiple viewport tags not supported

Previously, if you included multiple viewport tags in a web page, `WebView` would merge the properties from all the tags. In the new `WebView`, only the last viewport is used and all others are ignored.

## Default zoom is deprecated

The methods `getDefaultZoom()` and `setDefaultZoom()` for getting and setting the initial zoom level on a page have are no longer supported and you should instead define the appropriate viewport in the web page.

> **Caution:** These APIs are not supported on Android 4.4 and higher at all. Even if your `targetSdkVersion` is set to "18" or lower, these APIs have no effect.

For information about how to define the viewport properties in your HTML, read Pixel-Perfect UI in the WebView.

If you cannot set the width of the viewport in the HTML, then you should call `setUseWideViewPort()` to ensure the page is given a larger viewport. For example:

```
WebSettings settings = webView.getSettings();
settings.setUseWideViewPort(true);
settings.setLoadWithOverviewMode(true);
```

# Styling Changes

## Styling Changes

### The background CSS shorthand overrides background-size

Chrome and other browser have behaved this way for a while, but now `WebView` will also override a CSS setting for `background-size` if you also specify the `background` style. For example, the size here will be reset to a default value:

```
.some-class {
  background-size: contain;
  background: url('images/image.png') no-repeat;
}
```

The fix is to simply switch the two properties around.

```
.some-class {
  background: url('images/image.png') no-repeat;
  background-size: contain;
}
```

### Sizes are in CSS pixels instead of screen pixels

Previously, size parameters such as `window.outerWidth` and `window.outerHeight` returned a value in actual screen pixels. In the new `WebView`, these return a value based on CSS pixels.

It's generally bad practice to try and calculate the physical size in pixels for sizing elements or other calculations. However, if you've disabled zooming and the initial-scale is set to 1.0, you can use `window.devicePixelRatio` to get the scale, then multiply the CSS pixel value by that. Instead, you can also create a JavaScript binding to query the pixel size from the `WebView` itself.

For more information, see quirksmode.org.

### NARROW_COLUMNS and SINGLE_COLUMN no longer supported

The `NARROW_COLUMNS` value for `WebSettings.LayoutAlgorithm` is not be supported in the new `WebView`.

> **Caution:** These APIs are not supported on Android 4.4 and higher at all. Even if your `targetSdkVersion` is set to "18" or lower, these APIs have no effect.

You can handle this change in the following ways:

- Alter the styles of your application:

  If you have control of the HTML and CSS on the page, you may find that altering the design of your content may be the most reliable approach. For example, for screens where you cite licenses, you may want wrap text inside of a `<pre>` tag, which you could do with the following styles:

  ```
  <pre style="word-wrap: break-word; white-space: pre-wrap;">
  ```

  This may be especially helpful if you have not defined the viewport properties for your page.

- Use the new `TEXT_AUTOSIZING` layout algorithm:

  If you were using narrow columns as a way to make a broad spectrum of desktop sites more readable on mobile devices and you aren't able to change the HTML content, the new `TEXT_AUTOSIZING` algorithm may be a suitable alternative to `NARROW_COLUMNS`.

Additionally, the `SINGLE_COLUMN` value—which was previously deprecated—is also not supported in the new `WebView`.

## Handling Touch Events in JavaScript

If your web page is directly handling touch events in a `WebView`, be sure you are also handling the `touchcancel` event. There are a few scenarios where `touchcancel` will be called, which can cause problems if not received:

- An element is touched (so `touchstart` and `touchmove` are called) and the page is scrolled, causing a `touchcancel` to be thrown.

- An element is touched (`touchstart` is called) but `event.preventDefault()` is not called, resulting earlier enough that `touchcancel` is thrown (so `WebView` assumes you don't want to consume the touch events).