API Deprecations and Removals in Chrome 49

By Paul Kinlan

Paul is a Developer Advocate

In nearly every version of Chrome we see a significant number of updates and improvements to the product, its performance, and also capabilities of the web platform.

Deprecation policy

To keep the platform healthy, we sometimes remove APIs from the Web Platform which have run their course. There can be many reasons why we would remove an API, such as:

- They are superseded by newer APIs.
- They are updated to reflect changes to specifications to bring alignment and consistency with other browsers.
- They are early experiments that never came to fruition in other browsers and thus can increase the burden of support for web developers.

Some of these changes will have an effect on a very small number of sites. To mitigate issues ahead of time, we try to give developers advanced notice so they can make the required changes to keep their sites running.

Chrome currently has a process for deprecations and removals of API's, essentially:

- Announce on the blink-dev mailing list.
- Set warnings and give time scales in the Chrome DevTools Console when usage is detected on the page.
- Wait, monitor, and then remove the feature as usage drops.

You can find a list of all deprecated features on chromestatus.com using the deprecated filter and removed features by applying the removed filter. We will also try to summarize some of the changes, reasoning, and migration paths in these posts.

In Chrome 49 (Beta Feb 2nd, 2016. Estimated stable date: March 2016) there are a number of changes to Chrome

Use of "css" prefix in getComputedStyle(e).cssX is deprecated

TL;DR: The use of the "css" prefix in `getComputedStyle(e)` has been deprecated since it was not a part of the formal spec.

- Intent to Remove
- Chromestatus Tracker
- CRBug Issue

`getComputedStyle` is a great little function. It will return all CSS values of the DOM element's styles as they have been computed by the rendering engine. So for example, you could run `getComputedStyle(_someElement_).height` and it might return 224.1px because that is the height of the element as it is currently displayed.

It seems quite a handy API. So what are we changing?

Before the rendering engine of Chrome changed to Blink, it was powered by WebKit and that had let you prefix "css" to the start of a property. For example `getComputedStyle(e).cssHeight` instead of `getComputedStyle(e).height`. Both would return the same data as they mapped to the same underlying values, but it is this usage of the "css" prefix that is non-standard and has been deprecated and removed.

Note - `cssFloat` is a standard property and is not affected by this deprecation.

If you access a property this way in Chrome 49 it will return `undefined` and you will have to fix your code.

Use of initTouchEvent is deprecated.

TL;DR: `initTouchEvent` has been deprecated in favor of the `TouchEvent` `constructor` to improve spec compliance and will be removed altogether in Chrome 54.

Intent to Remove Chromestatus Tracker CRBug Issue

For a long time you have been able to create synthetic touch events in Chrome using the `initTouchEvent` API, these are frequently used to simulate Touch Events either for testing or automating some UI in your site. In Chrome 49 we have deprecated this API and will display the following warning with the intention to remove it completely in Chrome 53.

'TouchEvent.initTouchEvent' is deprecated and will be removed in M53, around September 2016. Please use the TouchEvent constructor instead. See https://www.chromestatus.com/features/5730982598541312 for more details.

There are a number of reasons why this change is good. It is also not in the Touch Events spec. The Chrome implementation of `initTouchEvent` was not compatible at all with Safari's `initTouchEvent` API and was different to Firefox on Android's. And finally, the `TouchEvent` constructor is a lot easier to use.

It was decided that we will aim to follow the spec rather than maintain an API that is neither specced nor compatible with the only other implementation. Consequently we are first deprecating and then removing the `initTouchEvent` function and requiring developers to use the `TouchEvent` constructor.

There **is** usage of this API on the Web but we know it is used by a relatively low number of sites so we are not removing it as quickly as we might normally. We do believe that some of the usage is broken due to sites not handling Chrome's version of the signature.

Because the iOS and Android/Chrome implementations of the `initTouchEvent` API were so wildly different you would often have some code along the lines of (frequently forgetting Firefox)

```
var event = document.createEvent('TouchEvent');

if(ua === 'Android') {
  event.initTouchEvent(touchItem, touchItem, touchItem, "touchstart", window,
    300, 300, 200, 200, false, false, false, false);
} else {
  event.initTouchEvent("touchstart", false, false, window, 0, 300, 300, 200,
    200, false, false, false, false, touches, targetTouches, changedTouches, 0, 0
}

document.body.dispatchEvent(touchEvent);
```

Firstly, this is bad because it looks for "Android" in the User-Agent and Chrome on Android will match and hit this deprecation. It can't be removed just yet though because there will be other WebKit and older Blink based browsers on Android for a while that you will still need to support the older API.

To correctly handle `TouchEvents` on the web you should change your code to support Firefox, IE Edge, and Chrome by checking for the existence of `TouchEvent` on the `window` object and if it has a positive "length" (indicating it's a constructor that takes an argument) you should use that.

```
if('TouchEvent' in window && TouchEvent.length > 0) {
  var touch = new Touch({
    identifier: 42,
    target: document.body,
    clientX: 200,
    clientY: 200,
    screenX: 300,
    screenY: 300,
    pageX: 200,
    pageY: 200,
    radiusX: 5,
    radiusY: 5
  });

  event = new TouchEvent("touchstart", {
    cancelable: true,
    bubbles: true,
    touches: [touch],
    targetTouches: [touch],
    changedTouches: [touch]
  });
}
else {
  event = document.createEvent('TouchEvent');

  if(ua === 'Android') {
    event.initTouchEvent(touchItem, touchItem, touchItem, "touchstart", window,
      300, 300, 200, 200, false, false, false, false);
  } else {
    event.initTouchEvent("touchstart", false, false, window, 0, 300, 300, 200,
      200, false, false, false, false, touches, targetTouches,
      changedTouches, 0, 0);
  }
}

document.body.dispatchEvent(touchEvent);
```

Error and success handlers required in RTCPeerConnection methods

TL;DR: The WebRTC RTCPeerConnection methods `createOffer()` and `createAnswer()` now require an error handler as well as a success handler. Previously it had been possible to call these methods with only a success handler. That usage is deprecated.

In Chrome 49 we've also added a warning if you call `setLocalDescription()` or `setRemoteDescription()` without supplying an error handler. We expect to make the error handler argument mandatory for these methods in Chrome 50.

This is part of clearing the way for introducing promises on these methods, as required by the WebRTC spec.

Here's an example from the WebRTC RTCPeerConnection demo (main.js, line 126):

```
function onCreateOfferSuccess(desc) {
  pc1.setLocalDescription(desc, function() {
     onSetLocalSuccess(pc1);
  }, onSetSessionDescriptionError);
  pc2.setRemoteDescription(desc, function() {
    onSetRemoteSuccess(pc2);
  }, onSetSessionDescriptionError);
  pc2.createAnswer(onCreateAnswerSuccess, onCreateSessionDescriptionError);
}
```

Note that both `setLocalDescription()` and `setRemoteDescription()` always had an error handler parameter, so simply specifying that parameter is a safe change.

In general, for production WebRTC applications we recommend that you use `adapter.js`, a shim, maintained by the WebRTC project, to insulate apps from spec changes and prefix differences.

Document.defaultCharset is deprecated

TL;DR: `Document.defaultCharset` has been deprecated to improve spec compliance.

Intent to Remove Chromestatus Tracker CRBug Issue

The `Document.defaultCharset` is a read-only property that returns the default character encoding of the user's system based on their regional settings. It's not been found to be useful to maintain this value because of the way that browsers use the character encoding information in the HTTP Response or in the meta tag embedded in the page.

By using document.characterSet you will get the first value specified in the HTTP header. If that is not present then you will get the value specified in the `charset` attribute of the `<meta>`

element (for example, `<meta charset="utf-8">`). Finally if none of those are available the `document.characterSet` will be the user's system setting.

Gecko has not supported this property and it is not cleanly specced so this property will be deprecated from Blink in Chrome 49 (Beta in January 2016). The following warning will appear in your console until the removal of the property in Chrome 50:

'Document.defaultCharset' is deprecated and will be removed in M50, around April 2016. See https://www.chromestatus.com/features/6217124578066432 for more details.

More discussion of the reasoning not to spec this out can be read on github https://github.com/whatwg/dom/issues/58

getStorageUpdates() removed

TL;DR: `Navigator.getStorageUpdates()` has been removed as it is no longer in the Navigator spec.

Intent to Remove Chromestatus Tracker CRBug Issue

If this impacts anyone I will eat my hat. `getStorageUpdates()` has hardly ever (if at all) been used on the web.

To quote the (very old version) of the HTML5 spec:

> If a script uses the `document.cookie` API, or the `localStorage` API, the browser will block other scripts from accessing cookies or storage until the first script finishes.
>
> Calling the `navigator.getStorageUpdates()` method tells the user agent to unblock any other scripts that may be blocked, even though the script hasn't returned.
>
> Values of cookies and items in the Storage objects of `localStorage` attributes can change after calling this method, whence its name.

Sounds pretty cool right? The spec even uses the word "whence" (which by happenstance is the only instance of whence in the spec). At the spec level there was a `StorageMutex` that controlled access to blocking storage such as `localStorage` and cookies, and this API would help free that mutex so other scripts would not be blocked by this `StorageMutex`. But it was never implemented, it's not supported in IE or Gecko, and WebKit's (and thus Blink's) implementation has been a no-op.

It's been removed from the specs for quite a while and has been removed completely from Blink (for the longest time it has been a no-op and did nothing even if called).

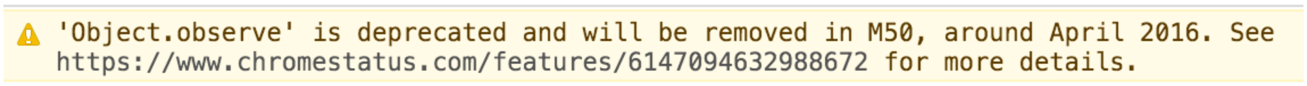
In the unlikely event that you had code that called `navigator.getStorageUpdates()` then you will have to check for the presence of the function before calling it.

Object.observe() is deprecated

TL;DR: `Object.observe()` has been deprecated as it is no longer on the standardization track and will be removed in a future release.

Intent to Remove Chromestatus Tracker CRBug Issue

In November 2015 it was announced that `Object.Observe` was being withdrawn from TC39. It has been deprecated from Chrome 49 and you will see the following warning in the console if you try to use it:



```
⚠ 'Object.observe' is deprecated and will be removed in M50, around April 2016. See
  https://www.chromestatus.com/features/6147094632988672 for more details.
```

'Object.observe' is deprecated and will be removed in M50, around April 2016. See https://www.chromestatus.com/features/6147094632988672 for more details.

Many developers liked this API and if you have been experimenting with it and are now seeking a transition path, consider using a polyfill such as MaxArt2501/object-observe or a wrapper library like polymer/observe-js.