

Respond to change with Object.observe



By Alex Danilo

Alex is a contributor to WebFundamentals

Lots of JavaScript frameworks using MVC or MDV need to respond to changes to the objects that model the state inside a web application. This capability is a fundamental part of a data-binding model.

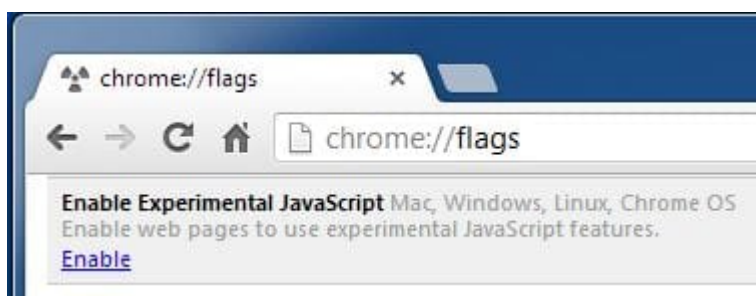
There are a number of different ways to monitor JavaScript objects and DOM properties to trigger some sort of action, but most of the techniques aren't ideal for various reasons such as performance, etc.

In order to improve the performance of web applications, a new method called Object.observe() has been proposed to TC39 - the standards body overseeing development of ECMAScript (JavaScript).

Object.observe() lets you add a listener to any JavaScript object that gets called whenever that object, or its properties, change.

You can try it out now in Chrome Canary version 25.

To experiment with this feature, you need to enable the *Enable Experimental JavaScript* flag in Chrome Canary and restart the browser. The flag can be found under 'about:flags' as shown in the image below:



Here's a simple example of how to set up an observer on an object:

```
var beingWatched = {};  
// Define callback function to get notified on changes  
function somethingChanged(changes) {  
    // do something  
}  
Object.observe(beingWatched, somethingChanged);
```



When the object 'beingWatched' is modified, it will trigger the callback function 'somethingChanged' which receives an array of changes that were applied to the object.

So the JavaScript engine is free to buffer up a number of changes and pass them all in a single call to the callback function. This helps with optimizing the callbacks so that your code can do lots of JavaScript manipulation but process only a few callbacks by batching the updates together.

The callback function will be triggered whenever a property is added, modified, deleted or reconfigured.

Another really nice thing when observing arrays is that if an array has had a number of changes made to it, you can use a [Change Summary](#) helper library to create a minimal change set, so that client side JavaScript doesn't have to manually scan the array to check each item.

You can iterate through each change quite easily, by doing something like the following in your 'somethingChanged' callback function:

```
function whatHappened(change) {  
    console.log(change.name + " was " + change.type + " and is now " + change.obj);  
}  
function somethingChanged(changes) {  
    changes.forEach(whatHappened);  
}
```

The *type* property identifies what happened to the object. Some examples of properties being set and the associated *type* can be seen in the code below.


```
beingWatched.a = "foo"; // new  
beingWatched.a = "bar"; // updated  
beingWatched.a = "bar"; // no change  
beingWatched.b = "amazing"; // new
```

The great thing about this technique is that all the monitoring smarts are inside the JavaScript engine which allows the browser to optimize it well and free your JavaScript up to implement functionality taking advantage of this feature.

Another really great feature for development is the ability to use `Object.observe()` to trigger the debugger whenever an object changes. To do that you'd use code something like the example below.

```
Object.observe(beingWatched, function(){ debugger; });
```

Here's a [great video introduction](#) about `Object.observe()` that explains it in detail.

There's also a [nice descriptive write-up available](#) and a [working example here](#) .

The TC39 standards body is seeking feedback on this feature, so go ahead and try it and let us know what you think.

Except as otherwise noted, the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#). For details, see our [Site Policies](#). Java is a registered trademark of Oracle and/or its affiliates.

Last updated July 2, 2018.