# Once Upon an Event Listener

**By** Jeff Posnick
Web DevRel @ Google

Pop quiz: what's the purpose of the third parameter passed to `addEventListener()`?

Don't be embarrassed if you thought that `addEventListener()` only took two parameters, or perhaps just always hardcode a value of `false`, with a vague understanding that it has something to do with… bubbles?

**Note:** The third parameter has historically been a boolean value, which defaulted to `false`, and which controlled whether the event listener was in capturing or bubbling mode.

## A more configurable addEventListener()

The `addEventListener()` method has come a long way since the early days of the web, and its new functionality is configured via a supercharged version of that third parameter. Recent changes to the method's definition allow developers to provide additional options via a configuration object, while remaining backward compatible when there's a boolean parameter or when an option is not specified.

We're happy to announce that Chrome 55 adds support for the `once` option in that configuration object, alongside the `passive` (implemented in Chrome 51) and `capture` options (implemented in Chrome 49). For example:

```
element.addEventListener('click', myClickHandler, {
  once: true,
  passive: true,
  capture: true
});
```

You can mix and match those options as appropriate to your own use case.

## The benefits of cleaning up after yourself

So that's the syntax for using the new **once** option, but what does that get you? In short, it gives you an event listener that's tailored to "one and done" use cases.

By default, event listeners persist after the first time they're called, which is what you want for some types of events—buttons that can be clicked multiple times, for example. For other uses, though, having an event listener stick around isn't necessary, and can lead to undesirable behavior if you have a callback that must only execute once. Hygienic developers have always had the option of using **`removeEventListener()`** to explicitly clean things up, following patterns like:

```
element.addEventListener('click', function cb(event) {
  // ...one-time handling of the click event...
  event.currentTarget.removeEventListener(event.type, cb);
});
```

The equivalent code, making use of the new **once** parameter, is cleaner, and doesn't force you to keep track of the name of the event (**`event.type`**, in the previous example) or a reference to the callback function (**`cb`**):

```
element.addEventListener('click', function(event) {
  // ...one-time handling of the click event...
}, {once: true});
```

Cleaning up your event handlers can also offer memory efficiencies by destroying the scope that's associated with the callback function, allowing any variables captured in that scope to be garbage collected. Here's one such example where it would make a difference:

```
function setUpListeners() {
  var data = ['one', 'two', '...etc.'];

  window.addEventListener('load', function() {
    doSomethingWithSomeData(data);
    // data is now part of the callback's scope.
  });
}
```

By default, the **`load`** event listener callback will remain in scope when it finishes running, even though it's never used again. Because the **`data`** variable is used inside the callback, it will also remain in scope, and never get garbage collected. If the callback were removed via the **once** parameter, though, both the function itself and anything that's kept alive via its scope will potentially be candidates for garbage collection.

## Browser support

Chrome 55+, Firefox 50+, and Safari's technology preview 7+ have native support for the **once** option.

Many JavaScript UI libraries provide convenience methods for creating event listeners, and some have shortcuts for defining one-time events—the most notable of which is jQuery's `one()` method. A polyfill is also available, as part of Andrea Giammarchi's `dom4` library.

## Thanks

Thanks to Ingvar Stepanyan for feedback about the sample code in this post.

---



*Last updated July 2, 2018.*