#SmooshGate FAQ

By Mathias Bynens
V8 JavaScript whisperer

What the *smoosh* happened?!

A proposal for a JavaScript language feature called **`Array.prototype.flatten`** turns out to be Web-incompatible. Shipping the feature in Firefox Nightly caused at least one popular website to break. Given that the problematic code is part of the widespread MooTools library, it's likely that many more websites are affected. (Although MooTools is not commonly used for new websites in 2018, it used to be very popular and is still present on many production websites.)

The proposal author jokingly suggested renaming **`flatten`** to **`smoosh`** to avoid the compatibility issue. The joke was not clear to everyone, some people started to incorrectly believe that the new name had already been decided, and things escalated quickly.

What does **`Array.prototype.flatten`** do?

`Array.prototype.flat`, originally proposed as **`Array.prototype.flatten`**, flattens arrays recursively up to the specified **`depth`**, which defaults to **`1`**.

```
// Flatten one level:
const array = [1, [2, [3]]];
array.flat();
// → [1, 2, [3]]

// Flatten recursively until the array contains no more nested arrays:
array.flat(Infinity);
// → [1, 2, 3]
```

Note: Since the publication of this article, **`flatten`** has been renamed to **`flat`** to resolve #SmooshGate. The above example has been updated accordingly, but keep in mind that the original name was **`flatten`**.

The same proposal includes `Array.prototype.flatMap`, which is like `Array.prototype.map` except it flattens the result into a new array.

```
[2, 3, 4].flatMap((x) => [x, x * 2]);
// → [2, 4, 3, 6, 4, 8]
```

What is MooTools doing that causes this problem?

MooTools defines their own non-standard version of `Array.prototype.flatten`:

```
Array.prototype.flatten = /* non-standard implementation */;
```

MooTools' `flatten` implementation differs from the proposed standard. However, this is not the problem! When browsers ship `Array.prototype.flatten` natively, MooTools overrides the native implementation. This ensures that code relying on the MooTools behavior works as intended regardless of whether native `flatten` is available. So far, so good!

Unfortunately, something else then happens. MooTools copies over all its custom array methods to `Elements.prototype` (where `Elements` is a MooTools-specific API):

```
for (var key in Array.prototype) {
  Elements.prototype[key] = Array.prototype[key];
}
```

`for-in` iterates over "enumerable" properties, which doesn't include native methods like `Array.prototype.sort`, but it does include regularly-assigned properties like `Array.prototype.foo = whatever`. But — and here's the kicker — if you overwrite a non-enumerable property, e.g. `Array.prototype.sort = whatever`, it remains non-enumerable.

Currently, `Array.prototype.flatten = mooToolsFlattenImplementation` creates an enumerable `flatten` property, so it's later copied to `Elements`. But if browsers ship a native version of `flatten`, it becomes non-enumerable, and *isn't* copied to `Elements`. **Any code relying on MooTools' `Elements.prototype.flatten` is now broken.**

Although it seems like changing the native `Array.prototype.flatten` to be enumerable would fix the problem, it would likely cause even more compatibility issues. Every website relying on `for-in` to iterate over an array (which is a bad practice, but it happens) would then suddenly get an additional loop iteration for the `flatten` property.

The bigger underlying problem here is modifying built-in objects. Extending native prototypes is generally accepted as a bad practice nowadays, as it doesn't compose nicely

with other libraries and third-party code. Don't modify objects you don't own!

Why don't we just keep the existing name and break the Web?

In 1996, before CSS became widespread, and long before "HTML5" became a thing, the Space Jam website went live. Today, the website still works the same way it did 22 years ago.

How did that happen? Did someone maintain that website for all these years, updating it every time browser vendors shipped a new feature?

As it turns out, "don't break the Web" is the number one design principle for HTML, CSS, JavaScript, and any other standard that's widely used on the Web. If shipping a new browser feature causes existing websites to stop working, that's bad for *everyone*:

- visitors of the affected websites suddenly get a broken user experience;
- the website owners went from having a perfectly-working website to a non-functional one without them changing anything;
- browser vendors shipping the new feature lose market share, due to users switching browsers after noticing "it works in browser X";
- once the compatibility issue is known, other browser vendors refuse to ship it. The feature specification does not match reality ("nothing but a work of fiction"), which is bad for the standardization process.

Sure, in retrospect MooTools did the wrong thing — but breaking the web doesn't punish them, it punishes users. These users do not know what a moo tool is. Alternatively, we can find another solution, and users can continue to use the web. The choice is easy to make.

Does that mean bad APIs can never be removed from the Web Platform?

It depends. In rare cases, bad features can be removed from the Web. Even just figuring out whether it's *possible* to remove a feature is a very tricky effort, requiring extensive telemetry to quantify how many web pages would have their behavior changed. But when the feature is sufficiently insecure, is harmful to users, or is used very rarely, this can be done.

`<applet>`, `<keygen>`, and `showModalDialog()` are all examples of bad APIs that were successfully removed from the Web Platform.

Why don't we just fix MooTools?

Patching MooTools so that it no longer extends built-in objects is a good idea. However, it doesn't solve the problem at hand. Even if MooTools were to release a patched version, all existing websites using it would have to update for the compatibility problem to go away.

Can't people just update their copy of MooTools?

In a perfect world, MooTools would release a patch, and every single website using MooTools would magically be updated the next day. Problem solved, right?!

Unfortunately, this is unrealistic. Even if someone were to somehow identify the full set of affected websites, manage to find contact information for each and every one of them, successfully reach out to all the website owners, and convince them all to perform the update (which might mean refactoring their entire code base), the entire process would take years, at best.

Keep in mind that many of these websites are old and likely unmaintained. Even if the maintainer is still around, it's possible they're not a highly-skilled web developer like yourself. We can't expect everyone to go and change their 8-year-old website because of a web compatibility issue.

How does the TC39 process work?

TC39 is the committee in charge of evolving the JavaScript language through the ECMAScript standard.

#SmooshGate caused some to believe that "TC39 wants to rename `flatten` to `smoosh`", but it was an in-joke that wasn't well-communicated externally. Major decisions like renaming a proposal are not taken lightly, are not taken by a single person, and are definitely not taken overnight based on a single GitHub comment.

TC39 operates on a clear staging process for feature proposals. ECMAScript proposals and any major changes to them (including method renaming) are discussed during TC39 meetings, and need to be approved by the entire committee before they become official. In the case of `Array.prototype.flatten`, the proposal has already gone through several stages of agreement, all the way up to Stage 3, indicating the feature is ready to be implemented in Web browsers. It's common for additional spec issues to come up during implementation. In this case, the most important feedback came *after* trying to ship it: the

feature, in its current state, breaks the Web. Hard-to-predict issues like these are part of the reason why the TC39 process doesn't just end once browsers ship a feature.

TC39 operates on consensus, meaning the committee has to agree on any new changes. Even if `smoosh` had been a serious suggestion, it seems likely that a committee member would object to it in favor of a more common name like `compact` or `chain`.

The renaming from `flatten` to `smoosh` (even if it hadn't been a joke) has never been discussed at a TC39 meeting. As such, the official TC39 stance on this topic is currently unknown. No single individual can speak on behalf of all of TC39 until consensus is reached at the next meeting.

TC39 meetings are generally attended by people with highly diverse backgrounds: some have years of programming language design experience, others work on a browser or JavaScript engine, and an increasing number of attendants are there to represent the JavaScript developer community.

How was SmooshGate resolved, eventually?

During the May 2018 TC39 meeting, #SmooshGate was officially resolved by renaming `flatten` to `flat`.

Last updated July 2, 2018.