A Detailed Introduction To Custom Elements

By Peter Gasston

March 4th, 2014 JavaScript, Techniques, Web Components

74 Comments

You've probably heard all the noise about Web Components and how they're going to change Web development forever. If you haven't, you've either been living under a rock, are reading this article by accident, or have a full, busy life that doesn't leave you time to read about unstable and speculative Web technologies. Well, not me.

Web Components are a suite of connected technologies aimed at making elements reusable across the Web. The lion's share of the conversation has been around Shadow DOM, but probably the most transformative technology of the suite is Custom Elements, **a method of defining your own elements**, with their own behavior and properties.

That's quite an ambiguous description, so the point of this article is to explain what Custom Elements are for, why they're so transformative and how to use them. Please note, first, that I'll talk about **custom elements** (common noun) when discussing the concept and **Custom Elements** (proper name) when discussing the technology, and secondly, that my humor tends to wear very thin very quickly. Let's push forward.

"What's The Point Of Custom Elements?"

The basic idea is that if you create an element that always performs the same role and has the same set of properties and functions applied to it, then you should be able to name it after what it does. We have the `video` element for displaying video, the `select` element for displaying a select box, the `img` element for displaying images (and saving us from typing two characters whenever we write it). A lot of elements describe their own function.

But **the Web today has to do a lot more work than it did previously**, and HTML can't always keep up with the rate of change. So Custom Elements are about giving us, the developers, flexibility to create elements based on their function and giving us low-level access to define their properties.

If the elements we create become well established, they could become a fully standardized part of a future HTML specification. The things we make could define the future of the things we make.

“But Can’t We Create Custom Elements Right Now In HTML?”

You’re right, notional reader, we can. It’s disgustingly easy. Just open your favorite text editor and make up an element in an HTML document, like so:

```
<apes>...</apes>
```

Open it in a browser. It works. You can style it, attach JavaScript events to it. It may not be “valid” (who cares about that these days, right, kids?), but it works. You can do it with any name you like, and it’ll create a new inline element.

Well, yes. Sure. You could certainly do that, and perhaps it would even make your markup a little more understandable to other people — but that’s really the only advantage it brings. Custom Elements are smarter than that, and they bring real, measurable benefits. We’ll get to the benefits of Custom Elements in a moment; first, I want to show how easy it is to make one.

“Are Custom Elements Easy To Create?”

They are, I just told you so in the previous sentence. The first step is to think

of a good name. The only rule here is that, to avoid clashing with current or future HTML elements, you must use a hyphen somewhere in the name. For example:

```
<great-apes>...</great-apes>
```

When you've decided on a name, the next step is to register it in the DOM, which is done by passing the name in as an argument in the JavaScript `registerElement()` method, like so:

```
document.registerElement('great-apes');
```

Now the DOM will recognize your newly registered `great-apes` element and the real fun can start. By the way, to confuse the terminology even further, an element created like this that's not defined in the HTML specification is known as a "custom tag," so don't be surprised if I use that term.

"I Still Don't Get What The Big Deal Is"

Bear with me, impatient notional reader. The big difference between puny custom elements and mighty custom tags (I hope you're not surprised by me using that term) is the interface that's exposed to the DOM. Custom elements, unregistered and unrecognized, use the `HTMLUnknownElement` interface, whereas registered and recognized custom tags use the `HTMLElement` interface.

What's the difference? With an `HTMLElement`, we can add our own methods and properties, creating essentially a per-element API. Wait, I understated how amazing that is: **a per-element API!!!** Yes, every custom tag can have its own API.

To initiate this, you would first define a new prototype, then attach your properties and methods to it. In this example, I'm creating a method named `hoot()` that logs a message to the console:

```
var apeProto = Object.create(HTMLElement.prototype);
apeProto.hoot = function() {
  console.log('Apes are great!');
}
```

The next step is to register the element, just as before, only this time adding an argument in the options of `registerElement()` to state that it should use our newly defined prototype:

```
document.registerElement('great-apes', {prototype: apeProto});
```

When this is done, you can query your element in the DOM and call the method:

```
var apes = document.querySelector('great-apes');
apes.hoot();
```

Now, this is the simplest example I could possibly think of, but just take a minute to consider how this could be extended further still: adding unique

properties, attributes and events to each element; putting markup in your element that renders with content passed in as attribute values; even having elements with no UI at all but that perform functions such as database queries. Honestly, the opportunity here is *huge*.

As a quick example of just how exceptionally useful Custom Elements can be, see Eduardo Lundgren's `google-maps` element, which embeds a Google Map and can have options passed in through attribute values, like this:

```
<google-maps latitude="-8.034881" longitude="-34.918377"></google-maps>
```

"Can Existing Elements Be Extended To Use This API?"

Wow, you really ask the most convenient questions. Yes, excitingly, we *can* make Custom Elements that extend existing elements. Yes, we can create a whole new API for existing HTML elements! I know, this sounds like the ramblings of a madman, right? But it's true!

As an example, let's create a table that has our `hoot()` method attached. To do this, we'd follow all of the steps in the previous section, then make the small addition of a new argument in the options of the `registerElement()` method, a lá:

```
document.registerElement('great-apes', {
  prototype: apeProto,
  extends: 'table'
```

```
});
```

The value of the `extends` argument informs the DOM that the custom element is intended to extend the `table` element. Now, we have to make the `table` element inform the DOM that it wants to be extended, using the `is` attribute:

```
<table is="great-apes">...</table>
```

The humble `table` element may now have its own API. For example, it could query its own data in a standardized interface. **A table that has an API to query its own data!!!** How can you not be excited by that?

For a real-world example of an extended element, take a look at Eduardo Lundgren's `video-camera`, which extends the `video` element to use live input from `getUserMedia()`:

```
<video is="video-camera"></video>
```

"OK, This Is Cool. What Else?"

A set of callback events (with brilliantly prosaic names) are fired throughout the lifecycle of Custom Events: when an element is created (`createdCallback`), attached to the DOM (`attachedCallback`) or detached from the DOM (`detachedCallback`), or when an attribute is changed (`attributeChangedCallback`). For example, to run an anonymous function each

time a new instance of a custom tag is created in a page, you'd use this:

```
apeProto.createdCallback = function () {...};
```

"How Do Custom Elements Work With Other Web Components Features?"

Custom Elements have been designed for complete interoperability with the companion features of the Web Components suite (and other generally related features). For example, you could include markup in the `template` element, which wouldn't be parsed by the browser until the element is initiated.

```
<great-apes>
  <template>...</template>
</great-apes>
```

You could ensure that the internal code is encapsulated from the browser and hidden from the end user with Shadow DOM. And sharing your element across multiple files and websites would be simplicity itself using HTML Imports.

If you're not familiar with any of these other technologies yet, don't worry: Custom Elements also work perfectly well on their own.

“Can I Use Custom Elements Today?”

Well, no. And yes. These aren’t just some pie-in-the-sky concepts; browser vendors are already working on them: the latest releases of Chrome and Opera have implemented the `registerElement()` method, and it also recently landed in Firefox Nightly. But, raw Custom Elements aren’t really ready for use in production just yet.

Gorillas are great apes… Look, it was either this or a screenshot of even more JavaScript code. (Image credits: Marieke IJsendoorn-Kuijpers)

However, there is a way around this, and that’s to use Polymer. In case you haven’t heard of it, it’s an open community project set up to make future

Web technologies usable today, and that includes Web Components and, through them, Custom Elements. Polymer is both a development library, which uses native implementations where available and polyfills where not, and a UI library, with common elements and patterns built using its own technology.

If you're at all interested in Custom Elements — and, as you've read almost to the end of this article, I'd suggest you probably are — then Polymer is your best option for learning and making.

"What About Accessibility?"

Ah, notional reader, here you have me. Using Custom Elements comes with one big caveat: **JavaScript is required**. Without it, your brand new element simply won't work and will fall back to being a plain old `HTMLUnknownElement`. Unless your element gets adopted natively by browsers, there's simply no way around this. Just plan for a graceful fallback, as you should be doing with JavaScript anyway.

As for further accessibility, it's really down to you. I strongly suggest adding ARIA roles and attributes to your custom elements, just as browser default UI elements have today, to ensure that everyone gets a first-class experience of them.

"Where Do I Go Next?"

Home, to have a good lay down. Or, if you prefer to carry on reading about Custom Elements, try some of these links:

- Polymer
  This the project that I talked about three paragraphs ago. Do you really need me to explain it again?
- Custom Elements
  This is a community-owned gallery of Web Components.
- "Custom Elements: Defining New Elements in HTML," Eric Bidelman, HTML5 Rocks
  Bidelman's article was invaluable to me in writing this piece.
- "Custom Elements," W3C
  The specification is fairly impenetrable, but maybe you'll get more out of it than I did.

(Huge thanks to Addy Osmani and Bruce Lawson for their feedback during the writing of this article.)

(al, il)

Front page image credits: Dmitry Baranovskiy.

Peter Gasston

Peter is a front-end technologist, developer, writer, speaker, and author of The Book of CSS3 and The Modern Web. He blogs at Broken Links and tweets as @stopsatgreen. He lives in London, England.

Related Articles

84

The Future Of Video In Web Design

84

Hidden Productivity Secrets With Alfred

84

What If Oscars Were Given To Movie Websites?

74 Comments

Rob Dodson March 6, 2014 9:10 am

1

Really great post Peter. One thing I want to point out, if you're extending a table element you'll also need to extend its prototype.

var apeProto = Object.create(HTMLTableElement.prototype);
document.registerElement('great-ape', {prototype: apeProto, extends: 'table'});

0

Ajay Zala March 6, 2014 8:04 am

2

Amazing Post... Although I have googled many designing sites but finding all in single post is time saving..really good job.

0

Bob March 6, 2014 7:29 am

3

Everyone who doesn't agree with me is wrong: this article was well written and amusing.

However... can't I just do all this anyway, with jQuery..?

Now.
Today.
Across all browsers.

0

Mark santiago March 6, 2014 6:12 am

4

Just read this last night and I was totally excited by the opportunities presented. As for the tone, i enjoyed it, no need to be so serious.This is cool stuff, let’s have fun with it!

0

ales March 6, 2014 5:36 am

5

Might as well just use angular.. it’s exactly what directives are anyway. I know having native APIs and stuff is cool, but think vanilla javascript vs. jquery or other libraries. Only few people would nowadays opt for pure javascript if they were to do some more advanced dom manipulation.

0

Stan March 6, 2014 12:49 am

6

This is so cool! Thanks for this article! :)

0

TJ March 5, 2014 6:54 pm

7

Would you recommend this be used on websites and web apps? I love the concept but this goes against performance a bit. To my understanding, the JavaScript used to make these elements would have to go in the head so that it is run before the browser finds the custom tag. This adds a network request (unless you inline it) and parsing of a script file before the browser can fully load the dom. In a game where milliseconds matter, I don't know that we can afford that loss, at least on a website.

1

Daniel Schwarz March 6, 2014 3:04 am

8

And especially on a mobile website.

0

Rob Dodson March 6, 2014 9:44 am

9

The JavaScript to register an element can come later in the document or be async. Before the JS has loaded the browser will regard any element it doesn't recognize as HTMLUnkownElement. So it won't break, it just won't

have any special abilities yet. Once the JS loads, all the elements will go through a process known as “upgrade”, and at that point they’re real custom elements. If you’re using a library like Polymer you’ll get a ‘polymer-ready’ event which informs you when this process is complete.

There is a small trade off because you’re loading something into your project that you didn’t hand code, but we do that all the time with javascript frameworks/libraries so I don’t really see the difference.

The real value comes in the fact that I can give you an element that: runs at 60fps, is totally responsive, works in *any* JS framework and has encapsulated CSS (if it’s using Shadow DOM). It also means that a really successful web component can be shown to the W3C, we can all agree that it rocks, and it can very quickly become a new standard in all the browsers (making it essentially zero cost for any site). The current practice of standardization involves theorizing about how we *think* an element should work, we debate it for years, and then maybe some syntax gets implemented that is a mixture of good and bad parts.

0

TJ March 6, 2014 12:42 pm

10

After doing some more digging, I found that you can declare your custom element asynchronously, however you then deal with FOUC (Flash of Unstyled Content).

You can fight FOUC with the :unresolved pseudo class in css to style a custom element that hasn’t been registered yet.

Here’s the article I got this from:

http://www.stevesouders.com/blog/2013/11/26/performance-and-custom-elements/

0

stephen March 5, 2014 4:59 pm 11

Great article! Thanks buddy

0

A. Henson March 5, 2014 12:05 pm 12

Nice article, love the writing style. I thought the humor was spot on, by the way, and even read several paragraphs to my team. Thanks for the post.

0

Sylvain Pollet-Villard March 5, 2014 11:54 am 13

Am I the only one here to think Web Components, and especially Shadow DOM,

are pure evil? I see Web Components as the official fig-leaf of “div soup jQueryUI-style™”, the kind of crappy DOM that ruins the semantics and accessibility of web pages.

We were close to a correct support of input type=”date” with optimized inputs and ergonomics for each device class, then an army of overexcited guys arrived yelling their super own datepicker Web Component rocks the house… of course, it looks nice and shiny and… very poorly suited to some atypical browsers / use cases. Oh, and about accessibility… well, no fucks given for people with disabilities. Bonus point : since custom elements are by definition custom, browser editors are completely helpless to try to repair the damage.

“Shadow DOM”, just the name should convince you that this is pure evil, standardized anti-standard…

1

14

Bob March 6, 2014 7:31 am

Agreed.

1

15

Marcy Sutton March 7, 2014 6:04 pm

What are you basing your accessibility assumptions on? I thought Custom Elements might be trouble for assistive technology as well, so I did some research…and I found that the Shadow DOM works just fine in a screen

reader.

You can read more here: http://substantial.com/blog/2014/02/05/accessibility-and-the-shadow-dom/

1

Sylvain Pollet-Villard

March 10, 2014 7:04 am

16

Technically, shadow DOM does not create new accessibility problems (except perhaps the need to have JavaScript enabled, but this is a global issue). However, I’m concerned about the fact that some developers systematically use poorly informative elements for their UI plugins, what I called “div soup jQueryUI-style™” even if jQueryUI is far from the worst example. They use div instead of semantic tags like lists (ul,ol,li) or headers (h1..h6) in order to minimize differences in interpretation between browsers and ensure that their own styles and behaviors are strictly respected. The tag information is useful, though not essential for screen readers who still have text, role, title and alt. But for other use cases like custom user stylesheets or virtual keyboards, it may become annoying. My concern is that shadow DOM could encourage this bad practice by hiding these presentational divs in the final DOM.

In short, as long as you use standard built-in elements and do not reinvent the wheel with Custom Events, everything will be fine. But I’m afraid this is not the case, considering the example of the date picker component compared to standard input type = “date”.

0

Timothy March 5, 2014 11:02 am

17

Several other programming platforms have also adopted this method of create new component types so that they can be reused in an application. Flex has has this for years in what they call MXML. Microsoft also uses this with XAML. And as some had mentioned, Microsoft had added something similar to this to earlier version of Internet Explorer. Unfortunately, it never really became part of the HTML standard.

Creating a component takes a little bit of work, but then it becomes reusable. This ultimately reduces the total amount of code in an application and makes the HTML look extremely clean.

Not only could you create components, but you could also create component containers to hold other components, such as, , or perhaps .

Also, think , , etc. There is so much potential here!

0

Tim March 5, 2014 7:00 am

18

These seem pretty cool.

I have such a pet peeve with coders and programmers writing elements with a lowercase and then uppercase letter, such as: registerElement
I know it’s somewhat standard to write things that way in code now, but why not just write it like RegisterElement? It is easier to read that way, especially if there is code in front of RegisterElement. Just a personal opinion, but it irks me when I see that and I cannot be un-irked.

0

TJ March 5, 2014 6:46 pm

19

Starting with a capital letter in a name tells a developer that it’s a class/constructor, for instance:

```
function CustomElement () {
...
}

var customElement = new CustomElement();
```

When that code is referrenced, we will know that customElement is the instance, while CustomElement is the class/constructor it came from.

0

Tim March 6, 2014 1:20 pm

20

Then why not use all lowercase for the instance?

1

Sean

March 24, 2014 11:20 am

21

If I see “alllowercase” instead of “allLowercase”, the first thing I would do is accidentally forget to retype the extra “L”, and the second thing I would do is wonder which case is an “allower”. Simply put, its confusing.

0

Ayyash

March 4, 2014 9:32 pm

22

I remember when IE4 adopted custom tags back in the day, you had to define the xml namespace in the html tag, and attach an HTC behavior through css, which by the way, I found brilliant , you have no idea how abusive I got with using them. The fact that you can wrap it all up in one tag and reuse, made application development much easier. This new standard is -I must say- neater, although the My:TagName of IE was better looking, but at least defining behavior in JS rather

than CSS is better. Today that is not very hard to do, all you have to do is create a DIV and attach some sort of behavior to it in JS. Two things thought:

– Accessibility: already mentioned.

– Abuse: I can only imagine the amount of HTML mark-up that will be injected through JS to make them look right. And poor next-developer, he/she has a mount of gibberish to sort through (think unobtrusive JS! this is a tiny bit better)

0

Ian Burns March 4, 2014 12:56 pm 23

Pitched exactly right – totally engaged me.

This has totally come out of the blue to me. Coming from a back-end development path, it is going to be marvellous to be able to apply that same level of control at the front end.

0

Steve March 4, 2014 12:35 pm 24

Does it remind anyone of HTML components (.HTC files) that Microsoft introduced back in Internet Explorer 5.5?

0

Ayyash March 4, 2014 9:36 pm

IE4 :)

0

sascha fuchs March 5, 2014 12:54 pm

Is not the first time that an old practice come back – remember the revival off the old new Box-Model with ‘box-sizing: border-box’ ;)

-1

burkanov March 4, 2014 12:19 pm

Sorry, but I still see no practical use at all. What are those things good for? I see no reason why someone should prefer custom elements over existing ones.

1

Matt March 4, 2014 2:29 pm

Not too many since you can just use CSS/jquery and use regular tags to make it as you wish…

0

Taufik Nurrohman March 4, 2014 5:11 pm

Regardless of any flexibility there, I feel these features are like **anti-standard** approach :(

0

Sean March 4, 2014 7:09 pm

The point of preferring the custom elements is pretty self-evident if you already have use for them.

If you had a `media-player` element, for example, you can define how it operates and how it initializes, and how it reacts to input, and how it deals with the server, one time, ever, providing attributes for configuration, much like they were constructor arguments.

With that in place you could just copy and paste that code from project to

project, and it would work. No worrying about how to bootstrap this whole module to the page and how to initialize it, no worrying about whether the bootstrap is taking place before/after the DOM is content-ready, or dealing with whether it should be a singleton or an instance, et cetera.

An IMG tag just works. You feed it one or a few attributes, and it does its job, and provides a semi-unique set of element properties in JS.

Don’t you occasionally wish you had tables that could .sort( ) or .filter( ) themselves?
Adding a new row of data would be as easy as calling myTable.insert(rowData), and it would automatically put itself in the right spot, sorted/filtered by the correct dimensions.
Or it could contain a summary-row, which automatically kept itself updated with a .reduce( ) algorithm, which was based on callbacks from rows being added/removed from the DOM (because the rows themselves might be simple “data-record” components.

Much like writing Angular directives, if you do it well, and you keep your code well-structured, you could then just drag and drop this anywhere, in the future. If everyone agrees to write these components in a highly self-contained way, and agrees to use vanilla JS and stable APIs, for the underlying implementations, then in the future, we have a zero-dependency set of infinitely-reusable community created components, which are all standards compliant, and require no more effort than including the JS *anywhere* in your distribution (with no effort to require or initialize or bootstrap), initializing is as easy as writing an HTML element, and you need to know none of the implementation, merely the interfaces provided for instantiation and interaction.

Personally, I consider that much, much nicer than reinventing the wheel, every time I want to create some carousel, or playlist.

2

Peter Gasston

March 5, 2014 1:42 am

31

Not much I can add to that. A great answer!

0

burkanov

March 5, 2014 4:48 am

32

Thank you, Sean
A very constructive answer.

0

Jonathan de Jong

March 6, 2014 11:08 pm

33

It’s times like these when I’d like an upvote button..
Just the fact that we could have advanced reusable elements independent of external libraries (I hate seeing a js folder with 30 files in it (for me as a developer. Minification, merging blabla)) is infinitely exciting in itself.

With a little imagination there’s no telling what this might result in.
Personally I’d want to play around with this and HTML5 games..

0

Jimmy Breck-McKye

April 9, 2014 2:04 pm

34

But can’t you do all that already with any kind of robust templating system?

I mean, I can already copy and paste a ‘media-slideshow’ tag in my source code today, binding up all my HTMLand attached JS, because I work with JSTL. That’s been around since, what, 2006?

This seems like the wrong solution to the right problem.

1

sunpietro

March 4, 2014 11:54 am

35

When people will start using custom elements, then the HTML will fail as XML did in some cases. There will be no standard. Everyone will have his/her own standard of custom HTML. That will be a disaster.

0

Alex Bondarev

March 6, 2014 12:40 am

36

That’s a good point. You also got me wondering about semantics. If I use `<article>` today it has a certain semantic meaning. If I use a `<blog-post>` tomorrow – what will happen to my semantics? Hm…

0

Alex

March 14, 2014 11:40 am

37

Good point indeed – Maybe its’ success lays in complicated custom apps/webapps etc.

At the moment i don’t really see benefit for ‘the normal’ web.

0

Adam

March 4, 2014 11:54 am

38

A big collection of custom elements can be found here:
http://gdriv.es/polymer/polymer-docs/

0

Brendan March 4, 2014 11:05 am

39

People are being too hard on you, man! Loved the article; will have to keep it in mind moving forward. Thanks for opening it up to me!

0

AdamG March 4, 2014 9:15 am

40

Informative, and as a smart-ass myself, your humor was not lost on me.
I think I have looked at Polymer a handful of times and did not see it’s total usefulness until today

0

Adam March 4, 2014 8:47 am

41

Exciting possibilities on the horizon. Great article. I enjoyed the humor.

0

Dean

March 4, 2014 8:09 am

42

Good article, a little bit different from other SM’s articles, but in a good sense ;-) However… I don’t really see a big advantage here…. Except direct function binding to the HTML elements, or am I mssing the point?

0

Peter Gasston

March 5, 2014 1:40 am

43

Direct function binding is a big advantage, definitely, but also being able to keep functionality contained within a single, appropriate tag, and giving existing elements an extended API.

Also, I only hinted at how this can be extended with other Web Components features, but when you see them all combined, you see the potential realised.

0

Jon Hobbs-Smith

March 4, 2014 8:03 am

44

Great article, and I didn’t find it condescending.

1

Farax March 4, 2014 8:02 am 45

Very nice. I think it is worth noting that you can also have custom attributes, as well as elements.

0

spencer March 4, 2014 7:45 am 46

Thanks for this. It is written in a way I can understand it – and I like your touch of humor too :)

0

Ben March 4, 2014 7:41 am 47

Funny and, my god, what a wealth of possibility this opens!

0

Drazen Mokic

March 4, 2014 7:32 am

48

Overall good info but i think you tried a little bit too hard to be funny at some point. No offense, you can’t please everyone anyway.

0

Uzair

March 4, 2014 6:33 am

49

Great article. And I don’t mind your sense of humor. :-)

0

David

March 4, 2014 6:27 am

50

It’s 2014. If you don’t have a javascript capable browser or have javascript turned off you don’t deserve graceful fallback. ;P

1

Dmitri

March 5, 2014 8:30 am

51

Nothing works with javascript off these days – agreed. I think the the reason we want to cut our reliance on it though is for SEO (I know bots can render HTML, but we are introducing some extra difficulties with bunch of code they have to crunch). Also, I have worked on too many projects with JS libraries all over the place, thousands upon thousands of lines of code in dozens of files. That is a total nightmare for me and for the poor hardware that has to deal with it for the inpatient user. Less is more.

I didn't mind the article style by the way – I have been very busy lately and literally just read the headlines here on SM but this one made me stop and admire the style a bit. I can see why some people might perceive it as condescending, which is OK. Better than sterile.

-3

Frontendjim March 4, 2014 5:45 am 52

Nice article. However, I find the way it's written a bit demeaning. Might just be me though. :)

-3

Peter Gasston March 4, 2014 6:04 am 53

I promise you that certainly wasn't the intention; I was aiming for 'funny', but

that can be very subjective…

6

Kriss Watt March 4, 2014 7:47 am

54

Seemed pretty on the money to me. Good read, thanks. I’m really excited to start building things using Custom Elements, even moreso after reading this and seeing Addy’s talk about Web Components at London JS Conf.

-2

Chris October 7, 2014 8:01 pm

55

I thought the humor made it much more fun to read. Don’t let the haters get you down ;) Thanks for the great information and a fun read.

3

Daniel Schwarz March 4, 2014 6:11 am

56

I found it a little condescending, and a bit silly. Not the usual standard of SM. But that aside, I did learn something new, so thanks.

0

deanomachino March 4, 2014 6:49 am 57

Agreed – useful article, but the tone is very condescending.

0

Rory March 4, 2014 8:44 am 58

whooooooosh!

0

↑ Back to top

With a commitment to quality content for the design community.
Founded by Vitaly Friedman and Sven Lennartz. 2006-2014.
Made in Germany. ✎ Write for us – Contact us – Impressum.