A New Paradigm for the Open Web

JavaScript & HTML5 Now

O'REILLY®

Kyle Simpson

O'REILLY®

oscon

open source convention

JULY 16–20, 2012 | PORTLAND, OR

oscon.com

5 Days,
200+ Speakers,
100+ Technologies, and
3000+ hackers like you.

Now in its 14th year, OSCON is where all the pieces of the open platform come together. Unlike other conferences that focus entirely on one language or part of the stack, OSCON deals with the open source ecosystem in its entirety, exactly as you approach it in your work. Join us for the annual gathering of open source innovators, builders, and pioneers. You'll be immersed in open source technologies and ideas, rub shoulders with open source rock stars, be seriously productive, and have serious fun with 3000+ people just like you.

2012 OSCON Tracks

- Business
- Cloud
- Community
- Data
- Geek Lifestyle
- Healthcare
- Java and JVM
- Javascript and HTML5
- Mobile
- Open Edu
- Open Hardware
- Ops
- Perl
- PHP
- Programming
- Python
- Tools and Techniques
- UX

SAVE 20%

Use code **EBOOK**

O'REILLY®

JavaScript and HTML5 Now

Kyle Simpson

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

JavaScript and HTML5 Now
by Kyle Simpson

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Revision History for the :
See *http://oreilly.com/catalog/errata.csp?isbn=9781449339098* for release details.

ISBN: 978-1-449-33909-8
1337878589

Table of Contents

JavaScript and HTML5 Now 1
More than Ajax 1
What's in a name? 3
Open will win 4
What changes in HTML5? 5
Minor markup changes make big differences 5
What about JavaScript? 6
JavaScript, HTML5's good parts 7
Geolocation: Where am I? 7
The JavaScript rabbit hole 8
Web Workers get to work 9
Just over the horizon 10
Conclusion 11

JavaScript and HTML5 Now

HTML and JavaScript are staples of web technology; they've been the foundation of "the modern web" for well over a decade. HTML is how the browser shows us content, and JavaScript is how we go beyond simply reading to meaningful interaction with that content. Developers and users demand more and more of these intertwined technologies every day.

You can see the obvious power these technologies are bringing to countless Internet-connected devices worldwide on your smartphone, or in an email client running in your browser. From social games to productivity apps to geo-aware check-ins to billion-dollar photo sharing services, the immersive interactivity of the web is transforming how we perceive and relate, not only with computers, but ultimately with each other.

HTML5 and JavaScript push this further, adding new features that let you build more powerful applications with new kinds of interactivity.

More than Ajax

A few years ago, the rise of Ajax gave developers a whole new approach to building applications on the web. Taking advantage of the ability of JavaScript to modify the current page in the browser in response to information from a web server, without requiring the user to initiate a separate page load, Ajax unleashed unprecedented creativity and imagination, stretching the boundaries of the web to its edges and beyond. It's almost as if Ajax enlightened us that literally anything was possible on the web, and so we demanded everything, and more.

When Ajax arose, the browser landscape was dominated by only a few players. Netscape and Internet Explorer (IE) were the viewports through which we experienced the web. IE's dominance in the browser market gave Microsoft the drastic luxury of settling into their lead and virtually halting new discovery and experimentation.

What were the burgeoning web designers and developers to do when what they imagined for the web exceeded the cold reality of what was possible? The web "made a way", and that way was Flash. Flash, originally built in the mid-'90s, and eventually shepherded by Macromedia and then Adobe, had been growing slowly in popularity for several years. It was a technology that promised things in a browser that JavaScript and HTML alone simply could not do. The nail in search of a hammer had found its match.

Flash freed itself of the limitations of HTML and JavaScript by using a plugin, a special extension that must be installed in every single browser, to interpret Flash content, display it, and receive interaction from the user. It is, for all practical purposes, a "black box" dropped into an existing page, where what goes on inside the Flash content is often quite unlike (and unrelated to) what's going on around it in the page. Because the Flash plugin was ubiquitous enough to serve the majority of web users, truly interactive web content didn't need to be tied to the limitations of JavaScript and HTML, nor did we have to wait for the few browser gatekeepers to give us what we wanted.

If Microsoft was saying that the web had come as far as it ever needed to go, Flash challenged that assertion and continued to experiment and evolve. And the creators of ever-more complex web content simply changed their medium, but kept on fueling their imaginations. Flash became the go-to technology for any content/interactivity of any significant complexity. It cemented itself as the platform of choice for animations, games, and visual effects.

JavaScript was a second-class citizen in this world; the fallback we used only when we had to. Whatever JavaScript could do, Flash could do it, and much more. But not everyone jumped ship. JavaScript developers created popular frameworks the web knows today, like Prototype, jQuery, and Dojo. Mainly, though, these frameworks spent much of their early efforts just trying to smooth out and fix many shortcomings of JavaScript itself.

Microsoft did little over this period to improve JavaScript, but this left the door open for other browser vendors to join the mix. Mozilla Firefox rose from Netscape. Opera had a browser, and so did Apple in Safari. Eventually, even Google entered the field with their Chrome browser. More browsers meant new innovation in web technology, especially JavaScript.

The rediscovery of JavaScript also started the "cross-browser wars", an epic (and ongoing!) battle on the part of developers to write code that could somehow run predictably in an unpredictable browser world. These new options gave hope that JavaScript, the language that already ships natively in all browsers, could perhaps come back to life, and finally grow and improve despite its warts. This would take work, though; perhaps no other language has received so much negative attention for its warts.

However, JavaScript the language had strengths as well. JavaScript was in need of a marketing campaign, so to speak. Douglas Crockford's "JavaScript: The Good Parts," for all the controversy over the details, was in ways the marketing brochure that JavaScript needed to (re)capture developers' attention. Now we actually knew how to use just "the good parts" and avoid the pitfalls of the rest.

By 2010, Flash and JavaScript coexisted as two powerful choices in the web technology platform. Flash was not even the only plugin approach; Microsoft had their own answer in Silverlight. Increasingly, the question for developers became: “Do I rely on a plugin or not?” And for many, the answer was, “both.”

The plugin strategy failed in the long run, however. Apple’s iPhone, intentionally or not, signaled the beginning of the end for the plugin-based web technologies, because Apple would, bolstered by their mobile market dominance, reject these plugins outright.

Thus came the question heard ‘round the world: “If iPhone doesn’t support Flash, how will we continue to serve interactive content on the web?” The answer, as it turns out, had already been in the works for a couple of years by some of the “non-IE” browser vendors (including Apple). It may surprise you that the answer was not, at least not directly, JavaScript.

While JavaScript is capable of building very rich interfaces and interaction elements, just as its Flash counterpart was, many in the field felt like its complex code meant we were trying too hard to build our web interfaces. Delivering all the code needed to build these complicated elements on every page load lacks elegance and challenges performance.

Moreover, if you want to show a video embedded on a web page, how will you do that if you don’t have Flash? JavaScript is not capable of displaying the frames of a video file, because the underlying presentation layer, the HTML, doesn’t give us that ability. Flash, for better or worse, had given us a much more powerful “canvas” on which to paint, and we liked it. HTML was rather limited, by comparison.

The answer: HTML5.

What’s in a name?

“HTML5” has become as much a marketing buzzword as “Ajax” ever was. What started out as a technical term to refer to the next “version” of HTML (version 4 is what we’d had since before the IE6 plateau), “HTML5” is now a label that’s casually thrown around to describe almost anything new the web platform introduces.

The term “HTML5” is as much an empty bullet point your boss might list on a quarterly company strategy slide as it is an actionable goal your development team can achieve. The term certainly catches attention (version 5 must be better than version 4!), and it drives sales of books, magazines, software, and even ads. I should know: I’m the co-author of “HTML5 Cookbook”. But let’s be honest: now HTML5 is basically an umbrella for everything, and thus it actually means nothing.

Look past the label, though, and you’ll see something more fundamental: the spirit of HTML5. Can we agree to just ignore the marketing buzz of “HTML5”?

What we are seeing with HTML5 will, a few years from now, be the obvious second major inflection point in the evolution of the web platform. If Ajax took the web from its childhood to its teenage years, HTML5 is the web platform coming into young adulthood, full of both idealism and restlessness, grounded by experience and perspective.

HTML5 is the web finally coming of age.

HTML5 is a whole new set of presentation-layer capabilities built directly into the markup. Where HTML4 had an `<img>` tag for displaying a picture of your grampa's 80th birthday on your Facebook page, HTML5 adds the `<video>` tag that presents video in the page just as naturally as we could read text. Where Flash let us draw lines and shapes and move them around to a timeline animation, HTML5 gives us a blank `<canvas>` and invites us to draw and redraw directly onto the page, to our heart's content.

HTML5 brings what we once could only do with the help of plugins like Flash to native HTML and JavaScript environments.

Open will win

Flash had, at its peak, a more than 99% install base worldwide; there's no question it was ubiquitous. But Flash also had a serious flaw (beyond its hundreds of zero-day security breaches that Adobe is constantly patching). Web creators trusted the keys of the dream car, Flash, to a single driver, a for-profit company that could substantially decide both the direction and the velocity of the web platform. We gave them all the control. Flash succeeded, and quickly, not just because of what capabilities it gave the web, but because Flash didn't have any real competition.

Both Microsoft (which invented Ajax) and Adobe (whose Flash platform pioneered and informed most of what we now call HTML5) are valuable and integral parts of the web, both what it was and what it is now becoming. But the long-term health of the web, it seems, depends on them not being the only drivers.

We had no choice at the time, of course. If the web was going to survive, it had to move forward, even if we didn't like who was driving it or how they drove; some progress was better than none at all. Thankfully, the web community now sees, in HTML5, that with many drivers going in many different directions, the diversity of thought and experimentation will lead to a more robust and more impactful web platform.

The spirit of HTML5 is the embodiment of the open web. The web is open because anyone can contribute to its future substance and direction, and because everyone who uses it is a co-owner in that openness. The open web belongs to no one and to everyone; we have embraced that as our highest mission. May we never permit a closed-walled regression where the web loses its openness. We, the members of the global web platform community, must make sure that everyone who pushes and extends HTML5 does so in a way that keeps it open.

Take for example the recent efforts from Mozilla with their Boot2Gecko (B2G) mobile project. Mozilla is creating a fully open, web-stack powered, mobile OS platform. What that means is that instead of a for-profit company owning and controlling the software that powers mobile devices, the collective efforts of the broader web platform community will own and drive it, headed and supported by a non-profit Mozilla. Native mobile applications will be built using the same open-web technologies that web-delivered applications can and will use on the same devices.

More importantly, Mozilla is openly discussing and collaborating with the community and standards bodies to codify the extension APIs in JavaScript, which will allow the B2G operating system to communicate directly with the mobile device capabilities (camera, phone, vibration, etc.).

Did you catch that? The very heart of the mobile device vendor world is being directly challenged, not by another closed vendor, but by the collective power of the open web platform and community.

Microsoft is building HTML/JavaScript as a core part of the application stack in Windows 8. In the same way that B2G mobile apps will be built in HTML/JS, so too will many full desktop applications for Windows 8 and beyond. That's an exciting prospect. The openness of the web, the spirit of HTML5, insists that Microsoft do so in an open way. Will they? The spirit of HTML5 openness depends on it.

HTML5 is a revolution in how we think about the web platform. It embraces a spirit of openness, wherein we all own, participate, and benefit. It says that whatever we need, we can build into the platform itself. This is why HTML5 is far more than just a technology implementation; it's a fresh new paradigm.

What changes in HTML5?

Most of the changes in HTML5 are small additions with major impact. It's not just the HTML5 specification itself, however, but a set of related specifications. For convenience, the rest of this article will group these neighboring technologies under the term "HTML5 & Friends" (H5&F).

Minor markup changes make big differences

H5&F freshens up our markup, making it significantly cleaner. It abandons the unnecessary complications that its XHTML (HTML as XML, essentially) predecessor saddled us with. You can now write "`<br>`" instead of "`<br />`", and "`<img>`" instead of "`<img />`".

The markup simplification extends to tag attributes as well. `<script>` and `<link>` tags now assume their default "type" attribute value, so they can be much shorter. Speaking of simpler markup, the Doctype declaration at the top is now much cleaner: "`<!DOCTYPE html>`".

Next, HTML5 provides several new tags that give more semantic (aka, "self-descriptive") structure to our documents. There's now a `<nav>` element (instead of `<div`

`class=“nav”>`) for, unsurprisingly, navigation in your page. There’s `<article>` and `<section>` elements for ... well, you guessed it: grouping your content.

Why do semantic tags matter when you can always create your own semantics through class attributes? The web is becoming increasingly interconnected, and the more our systems can understand each other (tags are for systems, not humans!) and what the intent or meaning is, the more powerful and useful it will ultimately be for users. When a search engine scans your page, now it can have a much better idea of what it’s looking at when you semantically describe your content with better tags.

HTML5 provides many new and powerful form elements. For instance, sliders and date pickers now automatically support with a single tag/attribute value what used to take hundreds of lines of JavaScript to create custom elements. Again, HTML5 builds natively into the web platform the things we all find most commonly useful.

The `<video>` and `<audio>` tags let you embed video and audio players right in the page, without plugins. The `<video>` tag by default creates a visible rectangular box to display the video frames. The `<audio>` tag by contrast is invisible by default. And, both tags take the “controls” attribute, which if present will cause a set of player controls to appear.

Want to draw? HTML5 gives you `<canvas>`, which creates a blank rectangular region in which you can then draw any manner of graphic or bitmap image information, including charts, animations, photos with image filters applied, and even frame captures from a `<video>` element. Whatever you can draw in Photoshop, you can now draw directly onto the page with `<canvas>` instead of `<img>` and an external file. Of course, `<img>` is still quite useful for less interactive work.

What about JavaScript?

As an avid fan of JavaScript, I know in writing this article I have been feeling like JavaScript is getting the short end of the stick. Perhaps you’re wondering how JavaScript fits into the H5&F picture?

I’ve got good news: JavaScript is more central and relevant to the web platform than ever before. Sure, HTML5 gives us a bunch of improvements to markup, and with it all the raw tools we need in the presentation layer to replace everything we used to do with Flash. But, the truth is, without JavaScript, much of this functionality is rather ... limited.

The most powerful and interesting part of H5&F is that it fully embraces JavaScript as a core technology of the platform. The days of JavaScript being a second-class citizen, a toy language, or an auxiliary tool only useful for simple form validation are over. JavaScript unlocks the power of all the new functionality in H5&F. JavaScript is at the very `<center>` of the web platform now (pun intended!).

HTML5’s power is tied to JavaScript.

JavaScript, HTML5's good parts

JavaScript provides critical support to HTML5's multimedia and graphics support. The `<video>` and `<audio>` tags, as we said, can display a default set of controls for media playback. These controls can even be styled via CSS. Often, though, web applications will want to use one of these elements in a way that requires more direct control of the playback. So, we use JavaScript.

Not only can we do the obvious start, stop, seek, and rewind of the media playback programmatically, but we can also change out the media being played, listen for events such as when the media starts and ends, and check load progress, change the playback rate, control the volume, and capture frames of the video, among other capabilities. That's a pretty powerful little JavaScript API for a single HTML tag.

The most common use case for these tags' APIs is to create your own custom player controls, while keeping the default controls hidden. The default controls are generally suitable for most uses, but if you want full control, the JavaScript API is your friend.

Making the `<canvas>` element work almost entirely relies on the JavaScript API. That is, if you just drop in a `<canvas>` tag, it will take up space in your page, but it won't have any content displayed in it. You need the JavaScript API of `<canvas>` to perform all the useful tasks.

The `<canvas>` tag lets you define lines and shapes using various drawing commands, such as `lineTo()`, `rect()`, `arc()`, etc. Once you have a group of shapes, you actually draw (render) them using a `stroke()` or `fill()` command. You can apply a variety of styles to the shape path rendering, such as line width, color, style, fill color and style, etc. You can draw images, repeating patterns, and gradients onto your `<canvas>` as well.

Just as you can draw onto a `<canvas>`, you can also erase all or part of the drawing and draw again. In this way, `<canvas>` lets you create frame-based animations of your drawings. The `<canvas>` defaults to transparent where nothing has yet been drawn, so you can layer `<canvas>` elements on top of other elements in the page, even each other, to create advanced visual effects.

Common uses for `<canvas>` include charting, graphic animations, games, audio visualization, video effects, and much more.

Geolocation: Where am I?

H5&F recognizes the increasingly mobile nature of our browsers by supporting geolocation. Especially in the mobile world, it is very handy for the web application to have access to location information about where the user and device are located. The Geolocation API allows the page to ask the browser for the current geo-coordinates (if the user grants permission, of course). This lets an application tailor the behavior to be contextual based on a user's location.

For example, you might suggest nearby coffee shops or bars for a user, or figure out their timezone and give them a time-of-day-appropriate greeting, like "good morning."

Once you have the geo-coordinates (latitude and longitude), you can use a variety of mapping API services from providers such as Google Maps to interpret the coordinates and manipulate the data. For instance, you can calculate driving directions or distances, do a reverse lookup to find the nearest street address (and match it to a business name), or other such tasks.

Geolocation can also help you provide "geofencing" capabilities to your application. For instance, imagine your application collects and shares photos, and tags them with geographic information. Perhaps your users are happy to share this information whenever they are out and about, but if they're at home, they'd rather keep their exact home location private. You can allow users to set up a geofence, a geographic region they define on a map. As users move inside or outside of this region, behaviors are activated according to the user's preferences.

Augmented reality is another application of geolocation data. This essentially means you combine real-world location information (a map, or a video image being captured by the on-device camera) with virtual information (place or street names, public social information left by other users, etc).

Knowing a user's location opens up a whole range of possible enhanced functionality. This is mostly possible through the user manually entering their own location, but the user experience is so much more friendly if you can use the Geolocation API to grab their exact coordinates automatically.

The JavaScript rabbit hole

The most advanced features of H5&F rely on JavaScript to do most of their work, giving web applications capabilities developers have wanted for years.

H5&F provides storage capabilities in the user's browser through the localStorage and sessionStorage APIs, as well as (in some browsers) indexedDB or webSQL client-side databases. Need even more storage capability on the user's system? The Filesystem API allows you to create a virtual filesystem on the user's computer, and read/write files in it.

Speaking of file access, the FileReader API allows you to read the contents of any file the user sends to the application page. In previous iterations of HTML/JS, users could select a file using an `<input type=file>` box, but the contents of the file were not accessible to JavaScript; the data could only opaquely be sent to a server. Now, JavaScript can access the contents of a specified file directly, and use and manipulate the contents right in the page.

This makes it much easier to create image editors running in the browser. Instead of having to wait to send the image file to the server and then have it send the image back, you can immediately grab the image contents in the browser and display them to a user (using `<img>` or `<canvas>` tags, for instance).

Ever been annoyed at sites that use and abuse the #hash at the end of the URL in your address bar to store and navigate different application states? Those URLs are uglier and harder to understand and share. That technique, though, was just a hack to get around limitations with the old browser history system.

HTML5 provides a much more powerful history system and APIs, which gives control over the browser's address bar and the forward and backward navigation queue. You can now push "states" into the queue, and listen for an event when the queue is navigated by the users' forward and back buttons. This mechanism even lets you update the displayed URL in the address bar without the browser having to make a whole new page request.

For years, the only way to get drag-and-drop behavior in a web application was to emulate it using a series of complicated event trackings with mousedown and mouseup. Recognizing how important drag-and-drop (DnD) is to user experience, H5&F adds a native DnD API. The system publishes a variety of events when a DnD event starts and completes, and you can listen for and respond to those events in various elements on the page.

One use-case for this is dragging an image from one folder in a user's library to another. Another use is to allow a user to drag a file from their desktop and drop it into the application. This action is akin to the user picking that file from an `<input type=file>` control, and it gives you the same access to the file properties and contents.

Web Workers get to work

H5&F also includes Web Workers to help things move smoothly. Almost all web browsers operate a page by using a single system thread to handle all the page rendering and its JavaScript code execution. This can lead to issues where long-running JavaScript can create visual stutters on the page where the rendering is blocked by the code execution, and vice versa.

One solution is to create a Web Worker, which is a sandbox of JavaScript code that is fired up and executed in an entirely separate thread, making it truly parallel to the main UI thread of the page.

For instance, say you need to perform some complex calculation on a set of data, one that might take several seconds to complete. You definitely don't want to create a three-second delay where the page locks up for the user. Instead, you can put that code into a Web Worker and send the data to it and wait for a response. It's a little bit like sending that data off to a remote server, but it never leaves the user's system.

An emerging use case for Web Workers is to put complex game logic in a Worker, freeing up the UI thread for smooth game-play rendering.

Web Sockets (socket-based network connections) let JavaScript speak more freely, and they're often combined with Web Workers. JavaScript has, since the advent of Ajax, been able to send requests to the server and receive responses back. However, this is basically a single-channel communication, meaning that the client must always initiate, and the server must always respond.

Two-way (or full duplex, as it's called) communication allows the server and the client to act as both sender and receiver simultaneously. This means that a server can choose when it needs to send an update of data to a connected client, without having to wait for that client to request it. So-called "push notifications" have become essential in areas like email and chat/IM.

The way a browser can establish such a connection is now exposed to JavaScript, in the form of the Web Sockets API. A Web Socket is, as it sounds, a persistent socket connection between the client (JavaScript) and the server. By allowing these two to connect and stay connected and talk to each other frequently, real-time communication becomes possible.

Application uses for real-time socket communication include chat, live tech support, multi-user collaboration, gaming, and much more.

Just over the horizon

HTML5/H5&F is not a set-in-stone snapshot of current technology, but rather a continuum of technology advances toward open native functionality. There are many new capabilities that are still very early that will further enrich the HTML5/JS application arena.

For instance, peer-to-peer communication (that is, direct connections between two users without a server in-between) is coming. This will enable file sharing, video chats, and other exciting possibilities. Some browsers are starting to experiment with the PeerConnection API, and on top of that, WebRTC, to establish robust browser-to-browser communications.

A complex and powerful system called Web Intents (inspired by Intents in the Android mobile OS) is coming soon to a few browsers. Web Intents allows a web application to invoke a generic action (like "send" or "edit"), and provide some data for the action. If any other web (or desktop) applications have previously registered to be able to handle that action, then the browser will automatically prompt the user to invoke that registered application to handle the intent, and then when done return the result back to the original application. This technique will simplify and empower web service collaboration and interaction.

As mentioned above with Mozilla's Boot2Gecko, another big push coming in H5&F is (mobile) device APIs. For instance, imagine a web application running on a mobile phone being able to directly access the phone's camera, vibration, battery status, and other device capabilities. The goal is to create a simple and consistent API for each of these device functions that's directly accessible to JavaScript in web applications. This opens up all the capabilities of native applications to the web platform.

Device APIs won't be limited to mobile, however. HTML5 web applications running in desktop browsers will need to be able to access the webcam and microphone functionalities. Video teleconferencing directly with other web users without a service in-between (charging you money) becomes a very real possibility.

Browsers have something called the "Shadow DOM." In a nutshell this means the browser creates DOM elements for certain UI components (like a date picker, for instance), but hides those child DOM elements from the actual DOM you can interact with. You only see the outer element, and the shadow DOM hides the rest. Why is this important? Because one proposal currently working its way to fruition will allow web developers to create their own custom tags and custom UI components, complete with their own shadow DOM and behavioral implementation.

Conclusion

HTML5 represents the best of what the web platform can offer; it's the realization of the openness that the platform always needed and deserved. If you can ask for it, there's a great chance that an API already is in progress, or will be soon. HTML5 isn't just a pile of cool new functionality, but a whole new philosophy that empowers web applications.

We're just now scratching the surface of what's possible. Last fall, JavaScript's creator Brendan Eich said: "Always bet on JavaScript." HTML5 keeps making that bet more and more profitable.