

# Chrome Dev Summit: Open Web Platform Summary



By Sam Dutton

Sam is a Developer Advocate

## Blink

by Greg Simon & Eric Seidel

Blink is Chrome's open-source rendering engine. The Blink team is evolving the web and addressing the issues encountered by developers.

There have been a number of behind-the-scenes improvements started since our April launch.

First thing we did was to delete half our source, which we didn't necessarily need. We're still not done! And we're not doing this blind: code removal is based on anonymously reported aggregate statistics from Chrome users who opt in to reporting.

We publish a new developer API every six weeks: the same as Chrome's shipping schedule is.

One big change we made when we forked from Blink was to add an intents system: every time before we're going to change the web platform, we send a public announcement to Blink dev announcing our intent to add or remove a feature. Then we go off, and we code it! And then the very next day after the feature is checked in, it's already there shipping in our Canary builds. This feature is off by default, but you can turn it on using `about:flags`.

Then, on our public mailing list we announce an intent to ship.

At [chromestatus.com](http://chromestatus.com) you can see the [features](#) we've worked on, the features we've shipped, and those we're planning to deprecate. You can also check the [Chromium Releases blog](#) [↗](#), which has links to bugs and to our tracker dashboard.

Another big change is that we're removing WebKit prefixes. The intent is not to use Blink prefixes, but to have run-time flags (and not just compile-time flags).

[Android WebView](#) has been a big challenge – but [HTML5Test](#) shows that things are getting better. We're much closer to desktop in terms of having one set of web platform APIs everywhere (Web Audio is a great example of this!)

But how does the sausage machine work? Every single change we make to Blink is immediately run through over 30,000 tests, not to mention all the Chromium tests that run additionally later. We use 24 hour sheriffing, with thousands of bots, thousands of benchmarks, and systems that throw millions of broken web pages at our engine to make sure it doesn't fall over. We know that mobile is significantly slower, and this is something we're working hard to improve.

So what's new?

- **Web Components:** check out Eric Bidelman's talk!
- **Web Animations:** complex, synchronized, high performance animations that uses the GPU wherever possible
- **Partial Layout:** only compute what you need!
- **CSS Grid**
- **Responsive images:** srcset or srcN or ?
- Faster text autosizing, and consistent sub-pixel fonts
- Skia, the graphic system used by Blink, is moving from GDI to DirectWrite on Windows

We want to know what you have to say!

If you feel C++ in your blood and want to write C++ with us, all of our code is open. You don't have to tell anybody or evangelize to us. You can just simply post a patch or [file a bug](#)!

**Slides:** [Blink](#)

## Security

by Parisa Tabriz

More people are connected to the web today than ever before – and from more places.

We're connected with our laptops, phones and tablets, and probably soon enough with personal devices and accessories. We access the internet from untrusted and sometimes even hostile networks. With so much of our lives moving online, it's imperative we take steps to protect our data and our users' data.

Above all, as developers we need to understand the necessity and practicality of SSL.

What's SSL? It stands for Secure Sockets Layer, and it's a cryptographic protocol designed to provide communication security over the internet. It guarantees privacy, via encryption and integrity, to prevent snooping or tampering with your internet connection. SSL has its flaws, but it's the leading way – and really the only way – to ensure any kind of data communication security on the internet.

According to [SSL Pulse](#), a year ago we had about just under 15% of SSL adoption; we're now over 50% adoption.

Two acronyms:

- **TLS:** for most intents and purposes the same as SSL. To be precise, SSL 3.1 was renamed to TLS, and TLS is the IETF Standard name. But they're interchangeable!
- **HTTPS:** this is HTTP over SSL, just the layering of the security capabilities of SSL and standard HTTP. First the client–server handshake, using public/private key cryptography to create a shared key – which is used by the second part of the SSL protocol to encrypt communication.

Networking on the internet may feel safe, immediate and fast. It feels like we're talking directly to the website. But in reality, it's not a direct connection. Our communications go via a wifi router, an ISP, and potentially other intermediary proxies between your device and the website. Without HTTPS, all our communications is in plain text.

Trouble is, users rarely type in a full URL specifying HTTPS – or they click a link using HTTP. Worse, it's possible to mount a (wo)man-in-the-middle attack and replace HTTPS with HTTP. A tool called SSLstrip introduced in 2009 does just that. Firesheep, from 2010, just listened

to opened wifi networks for cookies being sent in the clear: that meant you could listen in on chat, or log in to someone's Facebook account.

But SSL is (relatively) cheap, fast and easy to deploy (check out [ssllabs.com](https://ssllabs.com) [🔗](#) and Ilya Grigorik's book High Performance Browser Networking). For non-commercial use, you can even get free certificates from [startssl.com](https://startssl.com)! Public Key Pinning is designed to give website operators a means to restrict which certificate authorities can actually issue certificates for their sites.

"In January this year (2010), Gmail switched to using HTTPS for everything by default. .. In order to do this we had to deploy no additional machines and no special hardware. On our production frontend machines, SSL accounts for < 1% of the CPU load, < 10 KB of memory per connection, and < 2% of network overhead...

If you stop reading now you only need to remember one thing: SSL is not computationally expensive any more."

– Overclocking SSL, Adam Langley (Google)

Lastly, a couple of bugs we see most commonly:

- **Mixed content:** sites that use HTTP as well as HTTPS. Your user will get annoyed because they have to click a permission button to load content. (Chrome and Firefox actually bar mixed content from iframes.) Make sure that all of your resources on an HTTPS page are loaded by HTTPS, by using relative or scheme-relative URLs for example `<style src="//foo.com/style.css">`
- **Insecure cookies:** sent in the clear via an HTTP connection. Avoid this by setting the secure attribute on cookie headers. You can also use a new "Strict Transport Security" header to require SSL Transport Security (HSTS).

## Takeaways

- If you care about the privacy and integrity of your users' data, you need to be using SSL. It's faster, easier, and cheaper than ever.
- Avoid common implementation gotchas, like mixed content bugs or not setting the right HTTP header bits.
- Use relative or scheme relative URLs.
- Check out some of the new cool stuff, like HSTS and cert pinning

**Slides:** [Got SSL?](#)

# Media APIs for the multi-device Web

by Sam Dutton & Jan Linden


Along with a proliferation of new devices and platforms on the web, we're seeing huge growth in audio, video and realtime communication. Online media is transforming the way we consume media of all kinds.

A UK government study found that 53% of adults 'media multi-task' while watching TV: using mobile devices to share and consume media. In many countries TV viewing is down and online viewing is up. In China, for example, in 2012 only 30% of households in Beijing watched TV, down from 70% in 2009. According to the W3C Highlights 2013, 'In the past year video-watching on mobile devices has doubled. This year in the US, the average time spent with digital media per day will surpass TV viewing. Viewing is no longer a passive act. In the US, 87% of entertainment consumers say they use at least one second-screen device while watching television.' According to Cisco 'video ... will be in the range of 80 to 90 percent of global consumer traffic by 2017'. That equates to nearly a million minutes of video every second.

So what do we have for web developers? An ecosystem of media APIs for the open Web: standardized, interoperable technologies that work across multiple platforms.

## Takeaways

- WebRTC provides realtime communication in the browser, and is now widely supported on mobile and desktop. In total there are already over 1.2 billion WebRTC endpoints.
- Web Audio provides sophisticated tools for audio synthesis and processing.
- Web MIDI, integrated with Web Audio, allows interaction with MIDI devices.
- The audio and video elements are now supported on more than 85% of mobile and desktop browsers [\[7\]](#).
- Media Source Extensions can be used for adaptive streaming and time shifting.

- EME enables playback of protected content.
- Transcripts, captions and the [track element](#)  enable subtitles, captions, timed metadata, deep linking and deep search.

**Slides:** [Media APIs for the multi-device Web](#)

---

*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.*