A

BROWSER SUPPORT

Since writing this book, much has changed in the browser market. The Chromium project, which the Chrome browser is based on, stopped using WebKit and created their own fork, called Blink. This in turn meant that Opera, who as the book went to press stated they would be dropping their own engine and basing themselves on Chromium, would also use Blink. Additionally, new versions of Internet Explorer and Safari (11 and 7, respectively) have been released.

Documenting feature implementation in browsers means aiming at a moving target, so the best I can do is take a snapshot. When considering whether to use one of the features in this book, always check the following sites for the most up-to-date information:

- HTML5 Please, *http://html5please.com/*
- The CSS3 Test, *http://css3test.com/*
- The HTML5 Test, *http://html5test.com/*
- Can I Use..., *http://caniuse.com/*

When I started this book midway through 2012, I took a gamble on which features I thought would be best to cover, including not only those that had already been well-implemented but also some that I thought stood a good chance of being implemented when the book went to print (or soon after). As I write this in late 2013, it seems that the pace of wider adoption has been slower than I anticipated for some of the features contained in IE10 (such as Grid Layout, Regions and Exclusions), but everything else is proceeding apace.

The Browsers in Question

Far too many browsers exist for me to provide a decent overview of feature support on each. Instead, in this appendix, I'll stick to the key modern desktop browsers—Chrome, Firefox, IE10 and above, and Safari—and their mobile equivalents, as I've done throughout this book.

As this book was going to press, Opera announced that they would be phasing out their own Presto rendering engine and future versions of the browser would instead use Chromium, the branch of WebKit that Chrome is also based on. That doesn't mean Presto will be going away in the short term—it's already embedded on many devices that don't tend to update, such as TVs and games consoles. In the longer term, feature support should be considered the same as Chrome, but I've kept it distinguished here for legacy support.

When discussing mobile browsers, I usually mean both smartphone and tablet and, more often than not, that means Safari mobile and the Android browser (although both are based on WebKit, there's quite a deal of variety between them). Firefox, Internet Explorer, and Opera use the same rendering engine across different platforms (although see the previous paragraph about Opera), so I'll only mention the mobile version of those browsers where any differences exist (which is not often).

When I refer to Android, I mean the stock browser that comes with many versions of the Android OS. Some versions of Android include the new mobile version of Chrome, which, like Firefox and Opera, can be considered more or less equivalent to its desktop sibling.

As I've mentioned before, there really is no substitute for testing on actual devices. If possible, you should create a device library or join one in your area; if that's completely out of the question, ask other developers for their experiences.

Enabling Experimental Features

Many browsers, especially Chrome and Firefox, are being much more cautious than they used to with regard to implementing experimental features. Where previously they would implement features with a vendor prefix and roll them out to all users, now they usually require that you explicitly enable certain features with a configuration flag.

In Firefox, you do this by entering **`about:config`** in the URL bar, at which point you'll see a message that warns you of the consequences of dabbling

in the browser's inner workings. If this doesn't deter you, you can find the feature you want and enable it before restarting your browser in order to gain access to the now-enabled feature.

In Chrome, the process is much the same, except that you enter `chrome://flags`, no warning message appears, and the features are usually enabled by toggling a link marked **Enable**.

Chapter 1: The Web Platform

Every major modern desktop browser comes with a set of developer tools that include a console (only Internet Explorer 7 and below don't have one). The situation on mobile and tablet is a bit more complicated: most browsers don't have developer tools by default, but they can be connected to their desktop equivalents for debugging, as explained "Test and Test and Test Some More" on page 19.

Chapter 2: Structure and Semantics

The newer HTML5 structuring elements appear in IE9 and above and all other major modern browsers. Discussion around some of these elements is still ongoing as I write this. A `main` element was recently added, which is natively supported in Firefox and Chrome, while other elements are at risk of being dropped.

Using the attribute-based accessibility and semantic extensions WAI-ARIA, microformats, RDFa, and microdata in any browser is completely safe. The microdata API is implemented in Firefox and Opera but has recently been dropped from Chrome due to a perceived lack of interest.

Data attributes are also supported in all browsers, although the API using `dataset` is not present in IE10 and below, or Android 2.3 and below. The jQuery method works cross browser.

Chapter 3: Device-Responsive CSS

As I write this, Media Queries are available in IE9 and above and all other major modern browsers. The media features related to device dimensions are the most widely implemented. The `resolution` media feature is in Chrome, Firefox, and IE10 and above, but not yet in Safari (despite being implemented in the WebKit engine at the end of 2012).

The `devicePixelRatio` attribute is in IE11 and all other modern browsers, including mobile versions. The `dppx` unit is implemented in Chrome and Firefox.

The `@viewport` rule is in old Opera, IE10 and above, and WebKit, using the vendor prefix of each. The `matchMedia` API is in IE10 and above and all other modern browsers, but not in Android 2.3 and below.

The CSS `box-sizing` property is in all browsers, although it requires a vendor prefix in Firefox and in Android 3.0 and below. Only Firefox supports the non-standard `padding-box` value. The `calc()` value function is in IE9 and above and in other modern browsers, although it requires a *`-webkit-`* prefix

in Safari 6 and below (Mac OS and iOS). It's not implemented in Android or Opera.

The viewport-relative length units—`vh`, `vw`, etc.—are in IE9 (with a few bugs) and IE10 and above, Firefox, and most WebKit browsers except Android, but they are not present in Opera. The `rem` unit is in IE9 and above and in all other modern browsers.

The `object-fit` and `object-position` properties are implemented in Opera only and marked as "at risk" in the spec so face an uncertain future, especially now that Opera is based on Chromium. This was one of the risks I took in the book which didn't pay off.

Chapter 4: New Approaches to CSS Layouts

The multi-column layout properties are implemented in IE10 and above and all other modern browsers. The use of vendor prefixes is required in all browsers but Internet Explorer and old Opera. Firefox lacks support for the `column-span` property. Only old Opera and IE10 and above support the `break-before` and `break-after` properties.

Flexbox is supported in all major browsers, but with caveats. In Safari (iOS and Mac OS) the *`-webkit-`* vendor prefix is required, and Safari 6 and below use a hybrid of the current syntax and an older one: the `justify-content` property isn't implemented, and it instead has the old `box-pack` property. This has been fixed in Safari 7.

IE10 also uses an outdated syntax, with the *`-ms-`* prefix; I recommend you read the documentation in the IE10 Guide for Developers (*http://msdn.microsoft.com/library/ie/hh673531%28v=vs.85%29.aspx/*) for detailed information. This is fixed in IE11, and the prefix has been removed.

Firefox has an unprefixed implementation but only supports single-line flexbox, so the `flex-wrap` and `flex-flow` properties are ignored.

The Grid Layout module has changed syntax again since the release of the book. As explained in Chapter 4, IE10 and above currently supports an older version of the syntax with the *`-ms-`* prefix. Apple announced that Safari 7 would have an implementation using a more up-to-date syntax, but that doesn't seem to be the case in the final release. The `grid-template` property is not currently implemented in any browser.

Chapter 5: Modern JavaScript

The `async` attribute is in IE10 and most other browsers besides Android versions 2.3 and below and old Opera. The `defer` attribute is the same but also has support at least back to IE8.

The `addEventListener()` method is in IE9 and above and all other major browsers, as is the `DOMContentLoaded` event.

Despite the uncertainty around existing patents, touch events are in Chrome, Firefox, Opera, Safari for iOS, and Android. IE10 and above has support for pointer events, such as `MSPointerDown`, which are vendor prefixed.

The `querySelector()` and `querySelectorAll()` methods are fully implemented in all modern browsers, from IE8 and above. The `getElementsByClassName()`

method is almost as well implemented, lacking support for IE8 only. The `classList` object is in IE10 and above and most other browsers bar Android 2.3 and below.

Chapter 6: Device APIs

The Geolocation API is in IE9 and all other major browsers. Device orientation is present in mobile WebKit browsers, Chrome, and Firefox mobile. Do bear in mind, however, that device APIs depend on certain functions being available on the phone; just because the Device Orientation API is implemented in a browser, it doesn't necessarily follow that the device has an accelerometer.

The Full Screen API is implemented in IE11 and all other modern browsers, all with appropriate vendor prefixes. Firefox's implementation differs from the spec, but rather than trying to explain that here, I refer you to the Mozilla Developer Network article, "Using Fullscreen Mode" (*http://developer.mozilla.org/en-US/docs/Web/Guide/API/DOM/Using_full_screen_mode/*) Firefox, WebKit and Blink, and IE11 support the `:full-screen` pseudo-class, again with respective prefixes.

The Vibration, Battery Status, and Network Information APIs are available in Firefox mobile only. Chrome 30 introduced implementation of Vibration, but it's currently flagged off by default. Despite support for the others apparently landing in WebKit throughout 2012, I can't find any working implementations.

The `getUserMedia()` method is implemented in old Opera, and in Firefox and Chrome with vendor prefixes (`mozGetUserMedia`, `webkitGetUserMedia`).

Web Storage is in IE8 and above and all other major browsers. The Drag and Drop API is partially supported in IE8 and IE9, and fully implemented in IE10 and above and in other major desktop browsers. Owing to its nature, it isn't supported in mobile browsers.

The File API is fully implemented in Firefox, Chrome, Safari (iOS and Mac OS), and Opera, and partially supported in IE10 and above and Android. The FileReader API is fully implemented in IE10 and above and all other desktop browsers, plus WebKit mobile browsers including Android version 3.0 and above.

Chapter 7: Images and Graphics

Some form of SVG support is present in IE9 and above, Android 3.0 and above, and all other major browsers. SVG filters are slightly more limited, being unavailable in Android and available with strong limitations in IE9 and above, although using SVG filters on HTML elements only works reliably in Firefox. The new CSS `filter()` function is implemented in Chrome and Safari with the *`-webkit-`* prefix. The use of fragment identifiers in SVG is only possible in IE10 and above and in Firefox.

Support for the `canvas` element is in IE9 and above and in all other major browsers. Firefox, Chrome, Safari (desktop), and Opera all have implementations of WebGL, although it's disabled by default in some browsers, notably Safari and Chrome for Android.

Chapter 8: New Forms

Levels of support for the various form elements, especially those with on-screen controls, vary wildly among browsers and are changing all the time. Rather than try to capture that here, see the HTML5 Test, which has the most comprehensive and up-to-date coverage. Using the new input types is generally considered safe, as the browser will fall back to the text type if a different value is not recognized.

The Form Validation API is present in IE10 and above and in all other major browsers. Safari supports the API but has no on-screen error notifications.

Chapter 9: Multimedia

The video and audio elements, along with their related APIs, are in IE9 and above and in all other major browsers, although with the caveat about supported file types discussed in Chapter 9. The `track` element is supported in the desktop versions of IE10 and above, Safari 6 and above, Chrome, and Opera; Chrome for Android is the only mobile browser to offer support. Media Fragments are implemented in Firefox and WebKit browsers.

The Web Audio API is experimentally implemented in Chrome and Safari (iOS and desktop), using the `webkitAudioContext()` constructor, and work is underway to bring it to Firefox. Of Web RTC, only the `getUserMedia()` method is currently supported, as mentioned in Chapter 6.

Chapter 10: Web Apps

Support for AppCache is present in IE10 and all other major browsers.

Chapter 11: The Future

Chrome offers the most support for the new Web Components features; it has implemented the Shadow DOM, created with the `createShadowRoot()` method, but it must be explicitly enabled. The `template` element is implemented in Chrome and Firefox, as is support for scoped styles and registration of custom elements using `document.register()`.

CSS regions are implemented in IE10 and above, Safari 7, and Chrome. All require vendor prefixes on the properties, and IE10 and above only allows content inside an `iframe` as the source.

Exclusions are available exclusively in IE10 and above, using the *`-ms-`* prefix.

The Feature Queries `@supports` rule and `CSS.supports()` DOM method are available in Chrome and Firefox.

Cascading Variables are implemented in Chrome only and must be explicitly enabled.