

APR 21ST, 2014

## Document Loading and DOM Lifecycle Events

Document loading and load events can be a bit confusing for newcomers. Multiple names for the same basic things, incompletely documented, ambiguously explained and those ever-present browser inconsistencies don't exactly help. I want to try and remedy this by providing a rundown of document loading lifecycle events.

### Document start

The document begins to load; HTML is tokenized and parsed. Scripts are *usually* downloaded, parsed and executed as they are encountered, and hold up the parsing of the document, unless they have the `async` or `defer` properties. Async and deferred scripts are given to another thread to handle whilst the 'main' thread goes through the HTML and all other scripts in source order. Newer browsers also perform speculative parsing, racing ahead in the HTML to see if there are any resources it can start downloading whilst the current script is being obtained and run.

Once the HTML is parsed the DOM has been built, the browser waits for all deferred scripts to finish downloading, then executes them in their source order.

As they'll be executing against a document structure that isn't ready, it's best for any scripts running in this phase to just perform data setup and attach callbacks to events that will fire when the document is more complete, either `window.onLoad` or, more commonly, `DOMContentLoaded`.

### DOMContentLoaded, aka 'jQuery ready()

Once the HTML is parsed, the document tree is built, and all non-async scripts have been executed (*including* scripts with `defer` [note 1]), the browser will typically fire the first lifecycle event, `DOMContentLoaded`. This can happen before any resources (images, iframes, stylesheets, fonts, frames) are loaded. Most developers know this event through jQuery's `ready()`, which lets you attach callbacks to be executed once the document is ready to read and manipulate.

`DOMContentLoaded` is a relatively recent standard, however, and as you might expect, legacy IE doesn't implement it. Internet Explorer 5-8 set `document.readyState="interactive"` and fire `readystatechange`. The timing isn't perfect, though, and can sometimes fire late, so jQuery uses a trick where it constantly attempts to scroll the document until that action no longer returns an exception – at this point, we know IE's DOM is stable. You can check this out in the appendix below, showing how `ready()` does its stuff.

There also seems to be a [bug](#) in Firefox currently where `DOMContentLoaded` can fire **before** all scripts with `defer` have been run. Hopefully it should be fixed soon.

When the event fires, its callback gets executed as you'd expect. If a lot of JS is attached, this can take a few hundred milliseconds, but threads downloading async scripts and images will continue doing their thing whilst this happens (though JavaScript won't be run until the callback finishes). If you're interested in timings, WebPageTest exposes it with the `domContentLoadedEventStart` and `domContentLoadedEventEnd` metrics.

Resources loaded in this phase will fire their own `load` events as they pop into place.

### window.onLoad, aka 'body.onLoad'

Once all resources are downloaded (including async scripts), we fire `window.onLoad` [note 2]. This is the granddaddy of load events, and was implemented as far back as the Netscape days. It's implemented pretty consistently across browsers, and is a fairly common metric for "loadedness" (because the UI's responsive, the images are loaded, and the browser busy indicator usually switches off). Note that 'all resources' doesn't include AJAX requests, and that dynamically added resources don't trigger re-firings of the `onLoad` event.

`Window.onLoad` and `body.onLoad` are aliases for the same event, and will overwrite each other. There seems to be some misinformation circulating that `body.onLoad` is an alias for `DOMContentLoaded`, but **this is incorrect**.

If a dependent resource does not load (e.g. an image request returns a 404), this will not stop the event firing.

### Summary

- If you just need the DOM to be in a legal state for your JS to manipulate, you can target `DOMContentLoaded` rather than `window.onLoad`. jQuery's "ready" makes it quite easy to bind a callback to this event, or some suitable alternative. If you don't want to use jQuery, you can implement a fallback that uses `readystatechange`.
- `DOMContentLoaded` will not wait for async scripts, so stuff attached to this event shouldn't have any async dependencies. You should be able to rely on dependencies from scripts with 'defer', Firefox's current bugs notwithstanding.
- If you have an async dependency, wait for `window.onLoad`
- If you're interested in the loading of a particular element (including a script or stylesheet), you can listen to load events on those elements alone.

### Notes

1. There is apparently a [bug](#) in Firefox where `DOMContentLoaded` can fire **before** all scripts with `defer` have been run.
2. Caveat: Unless we're IE6-9, in which case it turns out [we might fire the onLoad event before async scripts are downloaded](#). Typical.

### Appendix: jQuery's cross-browser 'ready' implementation

```
jQuery core
1 // Catch cases where $(document).ready() is called after the
2 // browser event has already occurred
3 if ( document.readyState === "complete" ) {
4     return jQuery.ready();
5 }
6
7 // Mozilla, Opera and webkit nightlies currently support this event
8 if ( document.addEventListener ) {
9     // Use the handy event callback
10    document.addEventListener( "DOMContentLoaded", DOMContentLoaded, false );
11
12    // A fallback to window.onload, that will always work
13    window.addEventListener( "load", jQuery.ready, false );
14
15    // If IE event model is used
16    } else if ( document.attachEvent ) {
17        // ensure firing before onload,
18        // maybe late but safe also for iframes
19        document.attachEvent( "onreadystatechange", function() {
20            if ( document.readyState === "complete" ) {
21                document.detachEvent( "onreadystatechange", arguments.callee );
22                jQuery.ready();
23            }
24        });
25
26        // If IE and not an iframe
27        // continually check to see if the document is ready
28        if ( ! document.documentElement.doScroll && window == window.top ) {function() {
29            if ( jQuery.isReady ) return;
30
31            try {
32                // If IE is used, use the trick by Diego Perini
33                // http://javascript.nwbox.com/IEContentLoaded/
34                document.documentElement.doScroll("left");
35            } catch( error ) {
36                setTimeout( arguments.callee, 0 );
37                return;
38            }
39
40            // and execute any waiting functions
41            jQuery.ready();
42        })();
43    }
```

Posted by Jimmy Breck-McKye • Apr 21st, 2014 • [DOM](#), [JavaScript](#), [events](#), [front-end](#)



« [Disabling Firefox Safe Mode](#)

[JavaScript curry](#) »

### Comments

6 Comments Jimmy Breck-McKye Login

Recommend 2 Share Sort by Best

Join the discussion...

Edi • a year ago  
Thank you for this in-dept and very enlightening article! One question that remains for me is the following:  
// and execute any waiting functions  
jQuery.ready();  
})();  
}

Posted by Jimmy Breck-McKye • Apr 21st, 2014 • [DOM](#), [JavaScript](#), [events](#), [front-end](#)



« [Disabling Firefox Safe Mode](#)

[JavaScript curry](#) »

### Comments

6 Comments Jimmy Breck-McKye Login

Recommend 2 Share Sort by Best

Join the discussion...

Edi • a year ago  
Thank you for this in-dept and very enlightening article! One question that remains for me is the following:  
@jimmy - Great article! Could you please explain about dominteractive and page ready as well? ...  
can we say that dominteractive means that user can interact with the page? If yes, then which event corresponds to page render?  
^ | v • Reply • Share

Adi Prasetyo • a year ago  
come on man, you are not lazy programmer, in fact you writing awesome article  
^ | v • Reply • Share

ALSO ON JIMMY BRECK-MCKYE  
I'm a Pair-programming Skeptic - Jimmy Breck-McKye  
2 comments • 9 months ago  
TheMeerkat — Completely agree. Preferably there should be automated formatting (just sharing IDE config if IDE supports formatting would do the job) and some extra ...  
Style Only Part of a Line Path With Raphael.js  
1 comment • 2 years ago  
Anton — Oh, thanks!!! That I was looking for!  
A Simple Group Checkbox for Knockout.js  
2 comments • 2 years ago  
Josh Habdas — I'm not quite sure what you're asking me here - are you asking how to implement a group checkbox that is only flagged when all sub-checkboxes ...  
Hello, Octopress! - Jimmy Breck-McKye  
1 comment • 2 years ago  
Josh Habdas — Was very pleased to see Octopress 3 was recently tagged. If you haven't been back recently check out the most recent, related posts on the Octopress site. ...

### Recent Posts

- [Slaughterhouse Five: Micro-review](#)
- [Another Day in the Death of America: Micro-review](#)
- [Politics and the English Language: Micro-review](#)
- [Review - Wolfenstein: The New Order](#)
- [The JavaScript Single Var Style Is an Antipattern](#)
- [Is Predator \(1987\) a Film About the Limitations of Masculinity?](#)
- [On the Word, 'Beauty'](#)

### GitHub Repos

- [octopress-parse](#)  
I knocked up some utils to parse Octopress post stats
- [trkl](#)  
Reactive microlibrary providing Knockout-style observables and computed for 401 bytes
- [react-playground-bar-charts](#)  
Toy repository for learning React.js
- [hands-free-website](#)  
Use speech synthesis and recognition APIs to build a hands-free website
- [premock](#)  
Store up commands for a function that doesn't yet exist, and run them when it does
- [raphael-paragraph](#)  
Add paragraphs of wrapped text in Raphael.js
- [react-synthesizer](#)  
A simple WebAudio API synthesizer with a basic React UI
- [snakeGame](#)  
W.I.P.
- [dynamicText](#)  
Simple lib for dynamic text utils in JavaScript
- [blogStyle](#)  
Old wordpress blog for breckmckye.com
- [ffvii\\_damage\\_calc](#)  
Final Fantasy VII Damage Calculator
- [@libreckmckye](#) on GitHub