Using Modern Browser APIs to Improve the Performance of Your Web Applications

Nic Jansma
@NicJ
//nicj.net

Who am I?

Nic Jansma

Microsoft Sr. Developer (2005-2011)

- Windows 7 & IE 9/10 Performance Teams

Founding member of W3C WebPerf WG

Founder of Wolverine Digital LLC

Developing high-performance websites, apps and games

nic@nicj.net @NicJ http://nicj.net
http://github.com/nicjansma
http://www.slideshare.net/nicjansma/

Performance Measurement (<=2010)

Server-side

- HTTP logs
- Server monitoring (cacti / mrtg / nagios)
- Profiling hooks

Developer

- Browser developer tools (Firebug / Chrome / IE)
- Network monitoring (Fiddler / WireShark)

Client-side / Real World

- Date.now() !?!?
- Client side-hacks (Boomerang)

State of Performance (<=2010)

- Measuring performance from the server and developer perspective is not the full story
- The only thing that really matters is what your end-user sees
- Measuring real-world performance of your end-users is tough
- No standardized APIs in the browser that expose performance stats
- Other client hacks exist (eg timing via Date.now()), but these are imprecise and not sufficient

WebPerf WG

- Founded in 2010 to give developers the ability to assess and understand performance characteristics of their applications

  *"The **mission** of the Web Performance Working Group is to provide methods to measure aspects of application performance of user agent features and APIs"*

- Collaborative effort from Microsoft, Google, Mozilla, Opera, Facebook, Netflix, etc

W3C WebPerf Goals

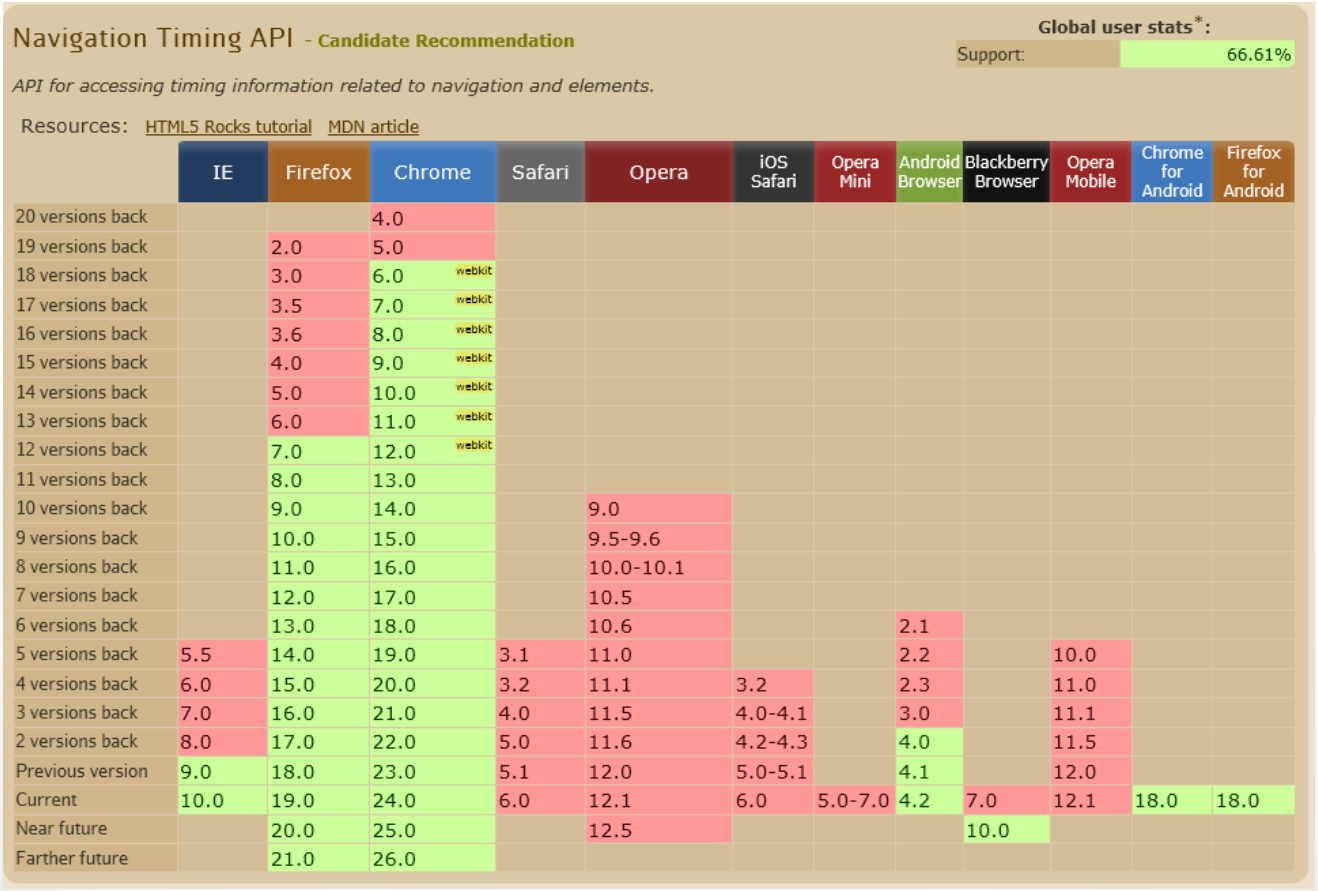
- Expose information that was not previously available
- Give developers the tools they need to make their applications more efficient
- Little or no overhead
- Easy to understand APIs

W3C WebPerf Specs

- Navigation Timing (NT): Page load timings
- Resource Timing (RT): Resource load times
- User Timing (UT): Custom site events and measurements
- Performance Timeline: Access NT/RT/UT and future timings from one interface
- Page Visibility: Visibility state of document
- Timing control for script-based animations: requestAnimationFrame
- High Resolution Time: Better Date.now()
- Efficient Script Yielding: More efficient than setTimeout(...,0) / setImmediate()

Navigation Timing (NT)

- http://www.w3.org/TR/navigation-timing/
- Page load and other network phase timings
- Great modern browser support



Navigation Timing API - **Candidate Recommendation**

API for accessing timing information related to navigation and elements.

Resources: HTML5 Rocks tutorial MDN article

Global user stats*: Support: 66.61%

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android
20 versions back			4.0									
19 versions back		2.0	5.0									
18 versions back		3.0	6.0 webkit									
17 versions back		3.5	7.0 webkit									
16 versions back		3.6	8.0 webkit									
15 versions back		4.0	9.0 webkit									
14 versions back		5.0	10.0 webkit									
13 versions back		6.0	11.0 webkit									
12 versions back		7.0	12.0 webkit									
11 versions back		8.0	13.0									
10 versions back		9.0	14.0		9.0							
9 versions back		10.0	15.0		9.5-9.6							
8 versions back		11.0	16.0		10.0-10.1							
7 versions back		12.0	17.0		10.5							
6 versions back		13.0	18.0		10.6			2.1				
5 versions back	5.5	14.0	19.0	3.1	11.0			2.2		10.0		
4 versions back	6.0	15.0	20.0	3.2	11.1	3.2		2.3		11.0		
3 versions back	7.0	16.0	21.0	4.0	11.5	4.0-4.1		3.0		11.1		
2 versions back	8.0	17.0	22.0	5.0	11.6	4.2-4.3		4.0		11.5		
Previous version	9.0	18.0	23.0	5.1	12.0	5.0-5.1		4.1		12.0		
Current	10.0	19.0	24.0	6.0	12.1	6.0	5.0-7.0	4.2	7.0	12.1	18.0	18.0
Near future		20.0	25.0		12.5				10.0			
Farther future		21.0	26.0									

NT: Why You Should Care

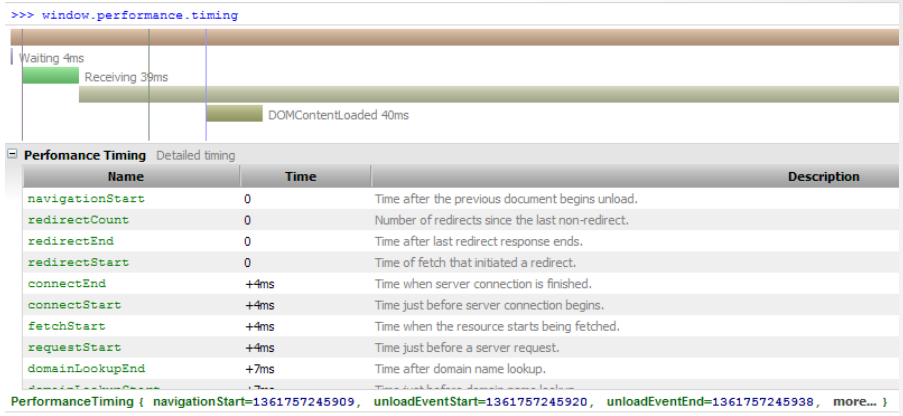
- How it was done before:

```
<html><head><script>
var start = new Date().getTime();
function onLoad {
    var pageLoadTime = (new Date().getTime()) - start;
}
body.addEventListener("load", onLoad, false);
</script>...</html>
```

- That's all you get: total page load time (kinda)
  - Technically, you get the time from the start of processing of JS in your HEAD to the time the body's onLoad event fires
- Says nothing of time spent before HEAD is parsed (DNS, TCP, HTTP request)
- Date.getTime() has problems (imprecise, not monotonically non-decreasing, user clock changes)

NT: How To Use

- DOM: window.performance.timing
- Phases of navigation
  - Redirect (301/302s)
  - DNS
  - TCP
  - SSL
  - Request
  - Response
  - Processing (DOM events)
  - Load



NT: How To Use

How to Use

- Sample real-world page load times
- XHR back to mothership

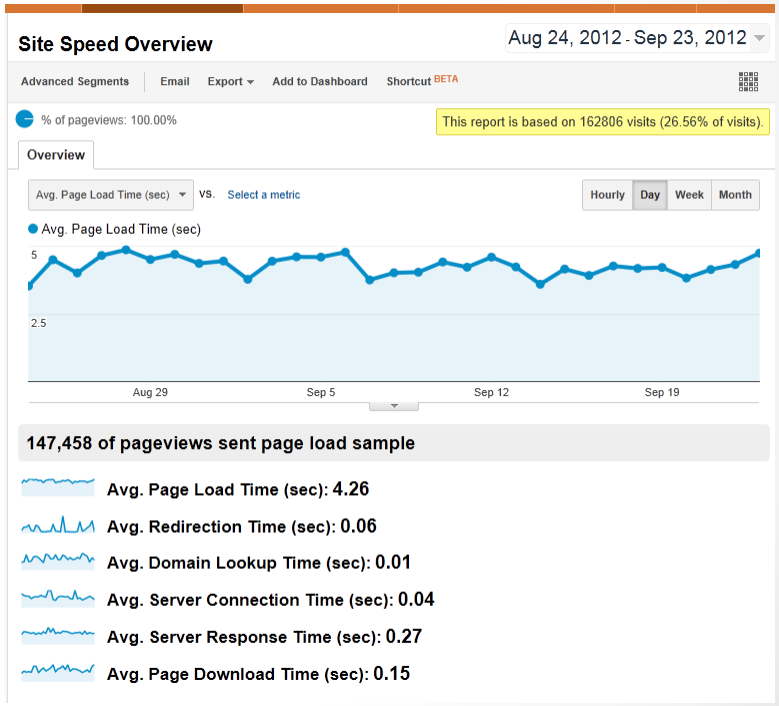
```
JSON.stringify(window.performance):

"{"timing":{"navigationStart":0,"unloadEventStart":0,"unloadEventEnd":0,"redirectStart":0,"redirectEnd":0,"fetchStart":1348506842513,"domainLookupStart":1348506842513,"domainLookupEnd":1348506842513,"connectStart":1348506842513,"connectEnd":1348506842513,"requestStart":1348506842513,"responseStart":1348506842595,"responseEnd":1348506842791,"domLoading":1348506842597,"domInteractive":1348506842616,"domContentLoadedEventStart":1348506842795,"domContentLoadedEventEnd":1348506842795,"domComplete":1348506842795,"loadEventStart":1348506842900,"loadEventEnd":1348506842900,"msFirstPaint":1348506842707},"navigation":{"redirectCount":1,"type":0}}"
```

Used by:

- Google Analytics' Site Speed
- Boomerang

Demo

- http://ie.microsoft.com/testdrive/Performance/msPerformance/Default.html



Resource Timing (RT)

- http://www.w3.org/TR/resource-timing/
- Similar to NavigationTiming, but for all of the resources (images, scripts, css, media, etc) on your page
- Get most of the data you can see in Net panel in Firebug/etc
- Support:
  - IE10
  - Chrome 25+ (prefixed)

RT: Why You Should Care

- How it was done before:

(it wasn't)

- For dynamically inserted content, you could time how long it took from DOM insertion to the element's onLoad event, but that's not practical for all of your resources

- You can get this information from Firebug, but that's not the end-user's performance

RT: How To Use

- DOM: See PerformanceTimeline
- Each resource:
  - URL
  - Initiator type (SCRIPT/IMG/CSS/XHR)
- Timings:
  - Redirect (301/302s)
  - DNS
  - TCP
  - Request
  - SSL
  - Response
  - Processing (DOM events)
  - Load

RT: How To Use

Gotchas

- Many attributes zero'd out if the resource is cross-domain (redirect, DNS, connect, TCP, SSL, request) **UNLESS** server sends `Timing-Allow-Origin` HTTP header

```
Timing-Allow-Origin: [* | yourserver.com]
```

- This is to protect your privacy (attacker can't load random URLs to see where you've been)

- Your own CDNs should send this HTTP header if you want timing data. 3rd-party CDNs/scripts (eg. Google Analytics) should add this too.

- Only first 150 resources will be captured unless `setResourceTimingBufferSize()` is called

Performance Timeline (PT)

- http://www.w3.org/TR/performance-timeline/

- Interface to access all of the performance metrics that the browser exposes (eg. Navigation Timing, Resource Timing, User Timing, etc)

- Support:
  - IE10
  - Chrome 25+ (prefixed)

PT: Why You Should Care

- Only way to access Resource Timing, User Timing, etc
- Gives you a timeline view of performance metrics as they occur
- Future interfaces (say, rendering events) can be added as long as they hook into the Performance Timeline interface

PT: How To Use

- performance.getEntries()
  - All entries in one array
- performance.getEntriesByType(type)
  - eg performance.getEntriesByType(“resource”)
- performance.getEntriesByName(name)
  - eg performance.getEntriesByName(“http://myurl.com/foo.js”)

```
interface PerformanceEntry {
   readonly attribute DOMString name;
   readonly attribute DOMString entryType;
   readonly attribute DOMHighResTimeStamp startTime;
   readonly attribute DOMHighResTimeStamp duration;
};
```

PT: How To USe

Example

User Timing (UT)

- http://www.w3.org/TR/user-timing/
- Custom site profiling and measurements
- Support:
  - IE10
  - Chrome 25+ (prefixed)

UT: Why You Should Care

- How it was done before:

```
<script>
var myMeasurements = [];
var startMeasure = new Date().getTime();
...
myMeasurements.push((new Date().getTime()) -
start);
</script>
```

- Problems: Date is imprecise, not monotonically non-decreasing, user clock changes

UT: How To Use

- Mark a timestamp:

```
performance.mark(“foo_start”)
performance.mark(“foo_end”)
```

- Log a measure (difference of two marks)

```
performance.measure(“foo”, “foo_start”, “foo_end”)
```

- Get marks and measures

```
performance.getEntriesByType(“mark”)
[
 {name: “foo_start”, entryType: “mark”, startTime: 1000000.203, duration: 0}
 {name: “foo_end”, entryType: “mark”, startTime: 1000010.406, duration: 0}
]

performance.getEntriesByType(“measure”)
[
 {name: “foo_end”, entryType: “measure”, startTime: 1000000.203, duration: 10.203}
]
```

UT: How To Use

- Easy way to add profiling events to your application
- Uses DOMHighResolutionTimeStamp instead of Date.getTime() for higher precision
- Can be used along-side NT and RT timings to get a better understanding of your app's performance in the real-world

Page Visibility (PV)

- http://www.w3.org/TR/2013/PR-page-visibility-20130219/

- Know when your application is ***not*** visible to the user

PageVisibility - **Candidate Recommendation**

Global user stats*: Support: 49.27%

JavaScript API for determining whether a document is visible on the display

Resources: MDN article

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android
20 versions back			4.0									
19 versions back		2.0	5.0									
18 versions back		3.0	6.0									
17 versions back		3.5	7.0									
16 versions back		3.6	8.0									
15 versions back		4.0	9.0									
14 versions back		5.0	10.0									
13 versions back		6.0	11.0									
12 versions back		7.0	12.0									
11 versions back		8.0	13.0									
10 versions back		9.0	14.0 webkit		9.0							
9 versions back		10.0 moz	15.0 webkit		9.5-9.6							
8 versions back		11.0 moz	16.0 webkit		10.0-10.1							
7 versions back		12.0 moz	17.0 webkit		10.5							
6 versions back		13.0 moz	18.0 webkit		10.6			2.1				
5 versions back	5.5	14.0 moz	19.0 webkit	3.1	11.0			2.2		10.0		
4 versions back	6.0	15.0 moz	20.0 webkit	3.2	11.1	3.2		2.3		11.0		
3 versions back	7.0	16.0 moz	21.0 webkit	4.0	11.5	4.0-4.1		3.0		11.1		
2 versions back	8.0	17.0 moz	22.0 webkit	5.0	11.6	4.2-4.3		4.0		11.5		
Previous version	9.0	18.0	23.0 webkit	5.1	12.0	5.0-5.1		4.1		12.0		
Current	10.0	19.0	24.0 webkit	6.0	12.1	6.0	5.0-7.0	4.2	7.0	12.1	18.0 webkit	18.0
Near future		20.0	25.0 webkit		12.5				10.0 webkit			
Farther future		21.0	26.0 webkit									

PV: Why You Should Care

- How it was done before:

(it wasn't)

or

"Are you still there?" popups

- There are times when you may want to know that you can "stop" doing something if the user isn't actively looking at your app:
  - Applications that periodically do background work (eg, a mail client checking for new messages)
  - Games (auto-pause)

- Knowing this gives you the option of stopping or scaling back your work

- Not doing background work is an efficiency gain -- less resource usage, less network usage, longer battery life

PV: How To Use

- document.hidden: True if:
  - User agent is minimized
  - Page is on a background tab
  - User agent is about to unload the page
  - Operating System lock screen is shown
- document.visibilityState:
  - hidden, visible, prerender, unloaded
- visibilitychange event
  - Fired whenever visibilityState has changed

PV: How To Use

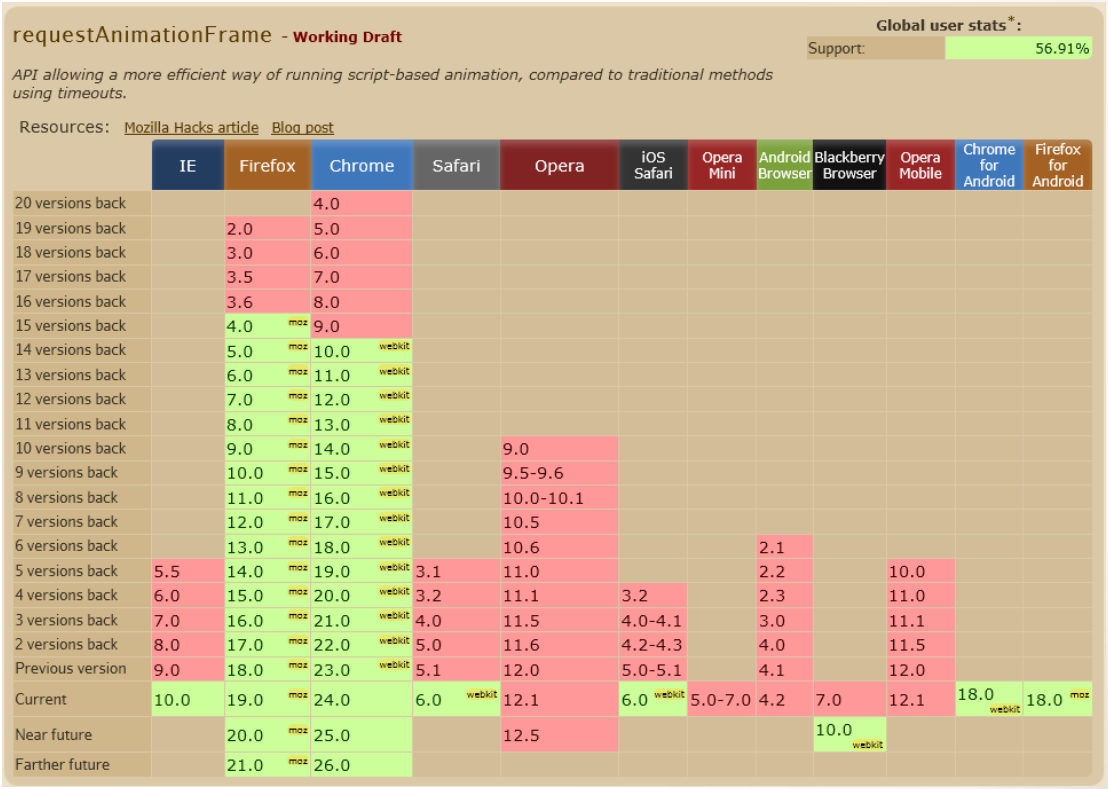
Automatically scale back checking for email if app isn't visible:

```
var timer = 0;
var PERIOD_VISIBLE = 1000;
var PERIOD_NOT_VISIBLE = 60000;

function onLoad() {
  timer = setInterval(checkEmail, (document.hidden) ? PERIOD_NOT_VISIBLE : PERIOD_VISIBLE);
  document.addEventListener("visibilitychange", visibilityChanged);
}

function visibilityChanged() {
  clearTimeout(timer);
  timer = setInterval(checkEmail, (document.hidden) ? PERIOD_NOT_VISIBLE : PERIOD_VISIBLE);
}

function checkEmail() { // Check server for new messages }
```

Timing control for script-based animations (requestAnimationFrame)

- http://www.w3.org/TR/animation-timing/

- Smarter animations



requestAnimationFrame - **Working Draft**

Global user stats*: Support: 56.91%

API allowing a more efficient way of running script-based animation, compared to traditional methods using timeouts.

Resources: Mozilla Hacks article, Blog post

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android
20 versions back			4.0									
19 versions back		2.0	5.0									
18 versions back		3.0	6.0									
17 versions back		3.5	7.0									
16 versions back		3.6	8.0									
15 versions back		4.0 moz	9.0									
14 versions back		5.0 moz	10.0 webkit									
13 versions back		6.0 moz	11.0 webkit									
12 versions back		7.0 moz	12.0 webkit									
11 versions back		8.0 moz	13.0 webkit									
10 versions back		9.0 moz	14.0 webkit		9.0							
9 versions back		10.0 moz	15.0 webkit		9.5-9.6							
8 versions back		11.0 moz	16.0 webkit		10.0-10.1							
7 versions back		12.0 moz	17.0 webkit		10.5							
6 versions back		13.0 moz	18.0 webkit		10.6			2.1				
5 versions back	5.5	14.0 moz	19.0 webkit	3.1	11.0			2.2		10.0		
4 versions back	6.0	15.0 moz	20.0 webkit	3.2	11.1	3.2		2.3		11.0		
3 versions back	7.0	16.0 moz	21.0 webkit	4.0	11.5	4.0-4.1		3.0		11.1		
2 versions back	8.0	17.0 moz	22.0 webkit	5.0	11.6	4.2-4.3		4.0		11.5		
Previous version	9.0	18.0 moz	23.0 webkit	5.1	12.0	5.0-5.1		4.1		12.0		
Current	10.0	19.0 moz	24.0	6.0 webkit	12.1	6.0 webkit	5.0-7.0	4.2	7.0	12.1	18.0 webkit	18.0 moz
Near future		20.0 moz	25.0		12.5				10.0 webkit			
Farther future		21.0 moz	26.0									

RAF: Why You Should Care

- How it was done before:

```
setTimeout(myAnimation, 10)
```

- Might be throttled in background tabs (Chrome 1fps)
- The browser can be smarter:
- Coalesce multiple timers (frame animations) so they all draw (and thus reflow/repaint) at the same time instead of odd intervals, along with CSS transitions and SVG SMIL
- Can sync with the device's frame rate

RAF: How To Use

- Instead of:

```
function render() { ... stuff ... }

setInterval(render, 16);
```

- Do:

```
// Find a good polyfill for requestAnimationFrame
(function animate() {
  requestAnimationFrame(animate);
  render();
})();
```

High Resolution Time (HRT)

- http://www.w3.org/TR/hr-time/
- A better Date.now
- IE10+, Chrome 23(?)+, Firefox 18(?)+

HRT: Why You Should Care

- Date.now() / Date().getTime() is the number of milliseconds since January 1, 1970 UTC.
- To be backwards compatible, modern browsers can only get as precise as 1ms
- Resolution of 15+ms in older browsers
- Is **not monotonically non-decreasing**: it does not guarantee that subsequent queries will not be negative. For example, this could happen due to a client system clock change.

HRT: How To Use

```
window.performance.now()
```

```
>>> performance.now()
3894956.5033731237
>>> performance.now()
4422789.271089402
```

- Monotonically non-decreasing
- Allows higher than 1ms precision
- Is defined as time since performance.timing.navigationStart
- NOTE: Is NOT milliseconds since UTC 1970

Efficient Script Yielding (setImmediate)

- http://www.w3.org/TR/animation-timing/
- Smarter than setTimeout(..., 0)
- Great demo @ http://ie.microsoft.com/testdrive/Performance/setImmediateSorting/Default.html

ESY: Why You Should Care

- How it was done before:

```
setTimeout(longTask, 0);
```

- Done to breakup long tasks and to avoid Long Running Script dialogs
- At max, setTimeout() in this manner will callback every 15.6ms (HTML4) or 4ms (HTML5) or 1s (modern browsers in background tabs) because callback depends on OS interrupts
- Setting a 0ms timeout still takes 4-15.6ms to callback
- Not efficient! Keeps CPU from entering low-power states (40% decrease in battery life)
- setImmediate yields if there is UI work to be done, but doesn't need to wait for the next processor interrupt

ESY: How To Use

```
setImmediate(longTask);
```

- Waits for the UI queue to empty
- If nothing in the queue, runs immediately (eg without setTimeout() 4ms/15.6ms/1s delay)

Questions?

@NicJ

nic@nicj.net

Slides @ http://www.slideshare.net/nicjansma/