HTML5 Revisited Workshop

Nathaniel T. Schutta
@ntschutta

HTML5 isn't the new kid on the block anymore.

But that doesn't mean it isn't interesting!

Browsers have made significant progress!

We can actually *use* a fair amount of the spec.

It will continue to evolve…

What is it?

HTML little long in the tooth...

And wasn't designed for applications.

Pushing boundaries.

De facto standards...

XHR anyone?

Standards aren't always clear cut.

Can contradict.

Not a conspiracy against developers.

Well not entirely.

Evolve over time.

A conversation.

Browser implementors, designers, standardistas.

Often a reaction to what we're doing.

We say we want standards...

Really want browser consistency.

Pain isn't standards, it's implementation.

2004 W3C workshop.

What should we do?

Evolving HTML lost.

Formed a new group.

WHAT Working Group.

Web apps!

Reversed engineered, and documented, parsing.

Web forms.

Canvas, audio, video tags.

Work continued
on XHTML 2.0.

You probably didn't notice.

WHAT WG had momentum.

W3C joined the effort.

Thus was born HTML5.

Since then…bit of a falling out between the groups.

It has now achieved "recommended" status!

http://www.w3.org/TR/html5/

W3C will publish snapshots of WHATWG work.

WHATWG will continuously update a "living standard".

https://html.spec.whatwg.org/multipage/

Curious about the process?

The Group That Rules the Web

http://www.newyorker.com/tech/elements/group-rules-web

So what is it again?

HTML5 is a collection of features...

Paving of cowpaths.

Evolutionary step.

What can I do?

Browser support isn't universal.

Shocking.

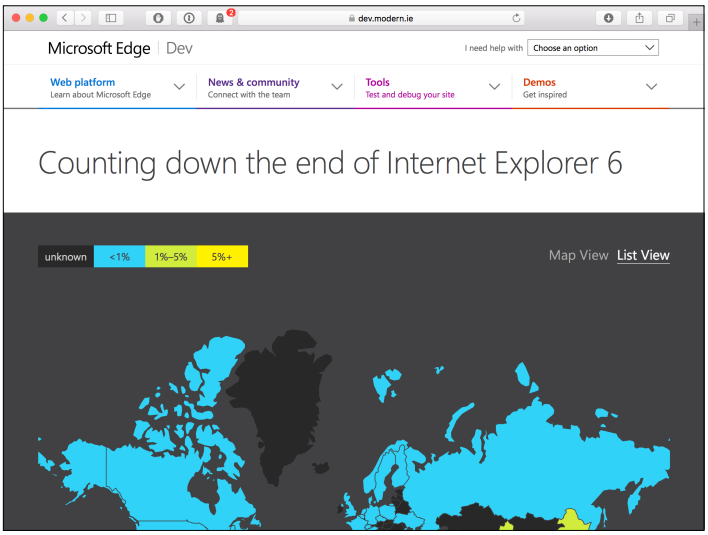
Older browsers pervasive.

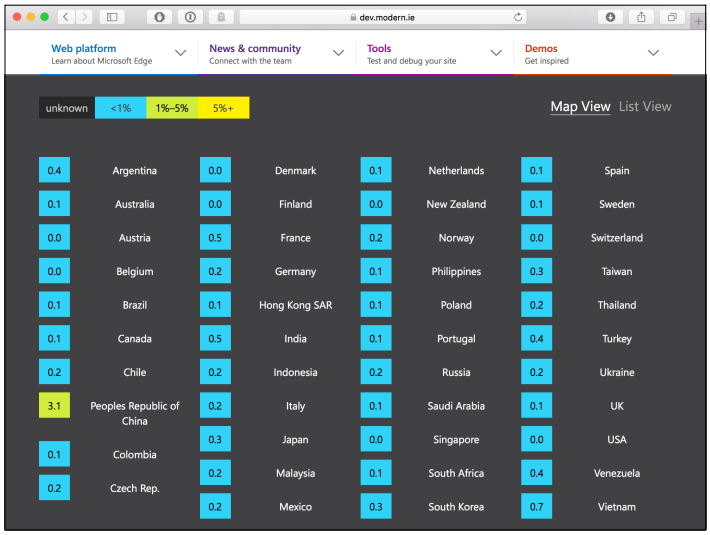
Check your logs. You'll probably be surprised.

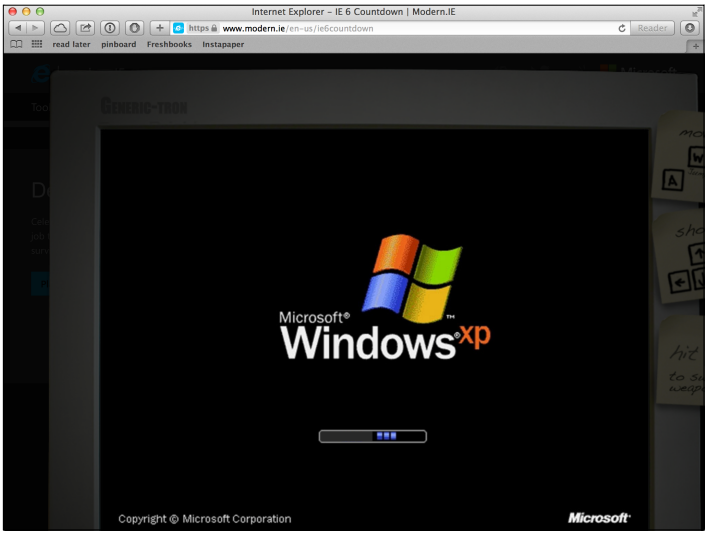
IE 6 is dead...

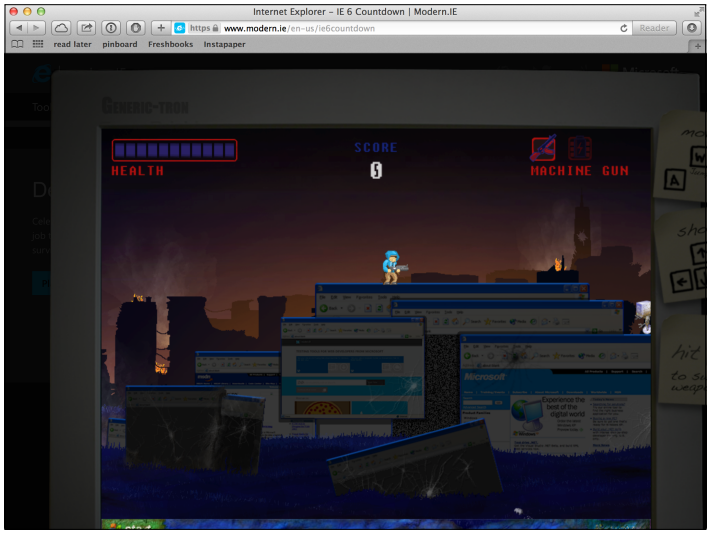
Even MS wants it gone.

https://dev.modern.ie/ie6countdown/









Agent sniffing?

NO!

Browsers lie...

This browser's user agent:
Mozilla/5.0 (Macintosh; Intel Mac OS X

This browser's user agent:
Mozilla/5.0 (Macintosh; Intel Mac OS X

This browser's user agent:
Mozilla/5.0 (Macintosh; Intel Mac OS X

This browser's user agent:
Mozilla/5.0 (Macintosh; Intel Mac OS X

And you can always mimic another user agent...

Feature detection.

The DOM is your answer key.

Modern browsers are evolving with spec.

Support is quite good.

Modernizer.

http://www.modernizr.com/

```
function supports_geolocation() {
    return !!navigator.geolocation;
}
```

```
function supports_canvas() {
    return !!document.createElement('canvas').getContext;
}
```

```
function supports_video() {
    return !!document.createElement('video').canPlayType;
}
```

```
function supports_input (input_type) {
    var input = document.createElement("input");
    input.setAttribute("type", input_type);
    return input.type !== "text";
}
```

Lab time!

Feature Detection - 2

- Using Modernizr, add the proper code to the four empty methods
- ${extract}/html5_workshop/labs/detection_modernizr.html

Not sure what your browser can do?

http://caniuse.com/
http://www.findmebyip.com/#target-selector

Another view...

Let's go!

Doctype.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
        "http://www.w3.org/TR/html4/loose.dtd">
```

Trans-what-now?

Quite a mouth full...

People designed expecting poor rendering.

Browsers standards support improved.

And they broke the web.

Quirks mode vs. standards mode.

And almost standards mode.

Seriously.

Let developers opt in.

HTML5 simplifies life.

```
<!DOCTYPE HTML>
```

It's almost too easy!

And you can type it.

Oh, must be the first line.

Blank line can kick you into quirks mode...

JavaScript is the default scripting language.

So long *type=*!

```
<script src="../javascript/fib.js"></script>
<script src="../javascript/jquery-2.1.4.js"></script>
<script>
$(document).ready(function(){
  $("#callFib").click(calculateFib);
  $("#callFibworker").click(calculateFibWW);
});
```

New semantic elements.

HTML5 adds new elements.

- section
- nav
- article
- aside
- summary/details
- header/footer
- time
- mark
- main
- figure/figcaption
- dialog

Defines things we've been doing for years.

With divs and ids.

It works, but lacks meaning.

Common markup.

Again, nod to what we're actually doing.

More meaningful than divs!

<time>

Encode time/date for machine use.

Meetings, birthdays, anniversaries ;)

```
<time datetime="2015-11-22">Today</time>
```

2 parts.

1. Machine readable.

```
<time datetime="2015-11-22">Today</time>
```

YYYY-MM-DD

Quite flexible.

https://html.spec.whatwg.org/multipage/semantics.html#the-time-element

Want time?

Add T, time in 24 hour, timezone offset.

```
e="2015-11-22T11:21:37-07:00">22 No
```

2. Human readable.

```
<time datetime="2015-11-22">Today</time>
```

Text doesn't have to match the datetime attribute.

It's human readable!

Next Sunday, tomorrow, in three days...

Could even be empty.

And if your browser doesn't support it?

Unknown elements rendered inline.

However, many of these elements are block.

In older browsers...

Style them as block.

HTML5 Reset.

http://html5doctor.com/html-5-reset-stylesheet/

Oh, before 9, IE won't style unknown elements.

Despite your CSS.

Also affects the DOM.

The workaround?

Create the element in JavaScript.

IE will allow you to style it.

Don't want to do that yourself?

No worries.

HTML5 enabling script.

https://github.com/aFarkas/html5shiv/

What's all the fuss about?

Divs work, right?

Document outline.

http://gsnedders.html5.org/outliner/

Before, headings were our only hope.

Sectioning content (article, aside, nav, section)...

Create new nodes.

Each has its own hierarchy.

Aids composability.

New Input Types.

We've spent a lot of time developing apps.

With a really limited palette.

Text box, text area, drop down, radio button...

Pretty limited.

Libraries help!

But why doesn't the browser do more?

Now it can!

HTML5 adds several new types.

And if your browser doesn't support it?

No worries.

Unknown types treated as text.

Even works in IE 6!

So what's been added?

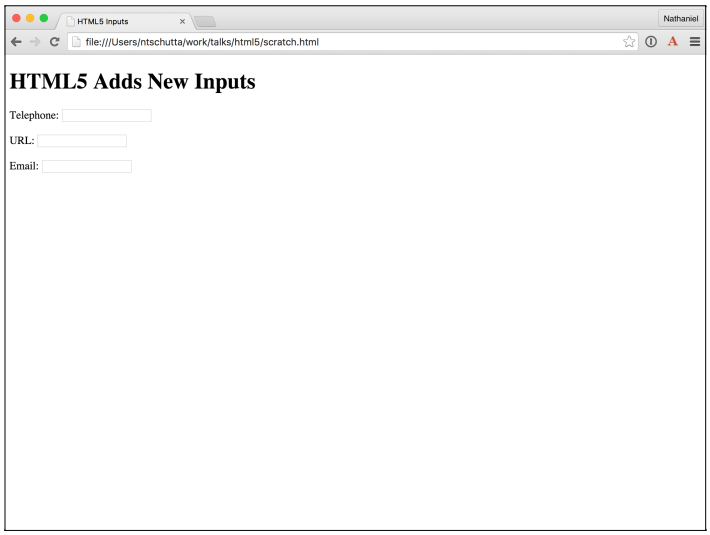
- search
- range
- number
- color picker
- telephone number
- url
- email
- date, month, week, timestamp
- datetime
- datetime-local

What do they do?

The spec doesn't say.

In many cases, they look just a text box.

For example...




```
<!DOCTYPE HTML>
<html>
<head>
    <title>HTML5 Input Types</title>
</head>
<body>
    <h1>HTML5 Adds New Input Types</h1>

    Telephone: <input type="tel"> <br/> <br/>
    URL: <input type="url"> <br/> <br/>
    Email: <input type="email"> <br/> <br/>

</body>
</html>
```

Impressed?

Yeah...

So what's the point?

What about your phone?

Does it have a physical keyboard?

Can't reconfigure a physical keyboard.

But when it's software...

Search.

Speaking of inputs that don't look much different...

HTML5 Adds New Inputs

Search:

HTML5 Adds New Inputs

Search:

HTML5 Adds New Inputs

Search:

HTML5 Adds New Inputs

Search:

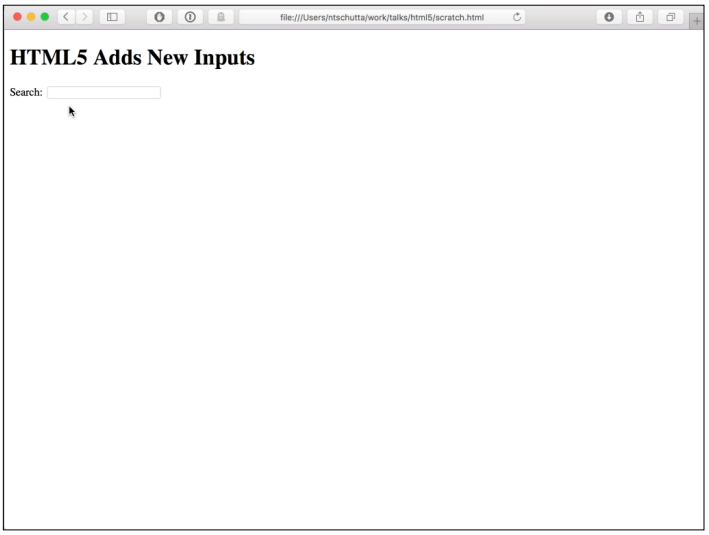
```
<!DOCTYPE HTML>
<html>
<head>
    <title>HTML5 Input Types</title>
</head>
<body>
    <h1>HTML5 Adds New Input Types</h1>

    Search: <input type="search" autofocus> <br/><br/>

</body>
</html>
```

Kind of rounded corners…

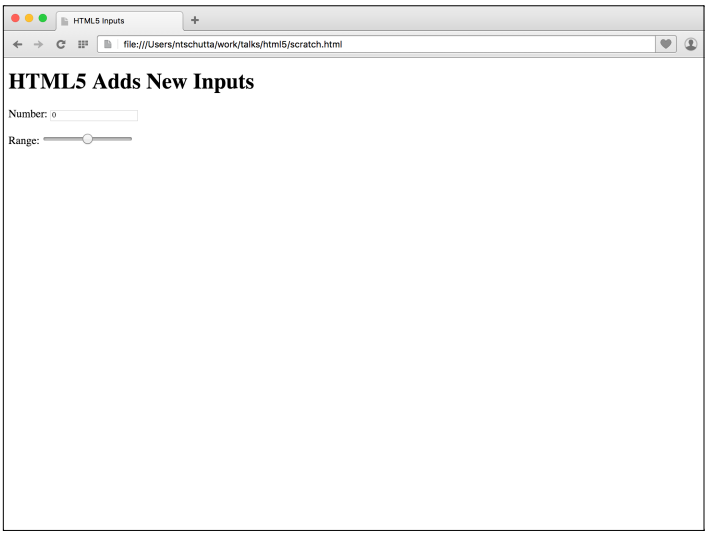
Oh an X...



Numbers.

Like email addresses & URLs, they're special.

How about spinners and ranges?




```
<!DOCTYPE HTML>
<html>
<head>
    <title>HTML5 Input Types</title>
</head>
<body>
    <h1>HTML5 Adds New Input Types</h1>

    Number: <input type="number"
                   step="2"
                   value="0"
                   min="0"
                   max="10"> <br/> <br/>

    Range: <input type="range"> <br/> <br/>

</body>
</html>
```

Attributes are optional.

Default step value is 1.

Ranges.

Slider control.

Shades of thick clients!

```
<!DOCTYPE HTML>
<html>
<head>
    <title>HTML5 Input Types</title>
</head>
<body>
    <h1>HTML5 Adds New Input Types</h1>

    Number: <input type="number"
                   step="2"
                   value="0"
                   min="0"
                   max="10"> <br/> <br/>

    Range: <input type="range"> <br/> <br/>

</body>
</html>
```

Also includes some handy JavaScript.

stepUp(n)
stepDown(n)

Increase/decrease the value by n.

valueAsNumber

Returns value as a number!

The value attribute
is a string...

Useful?

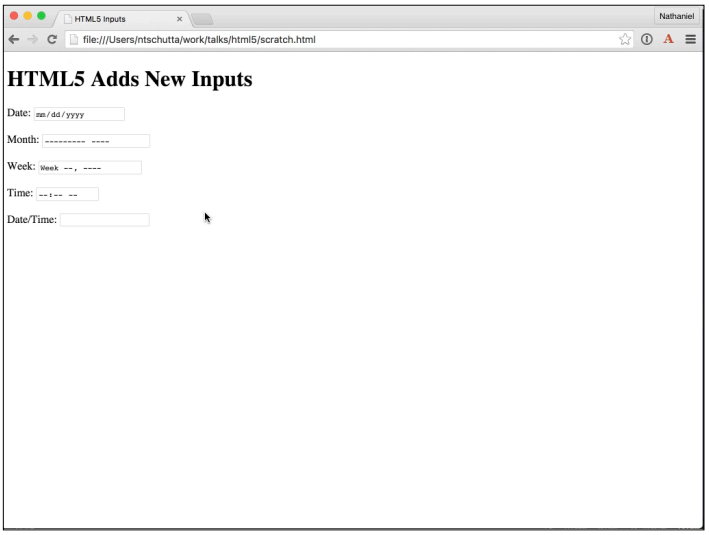
Date pickers.

Why don't we have a
native date picker?

Now we do!

But it's only in Opera and Chrome.

And you may not like it.




```
<!DOCTYPE HTML>
<html>
<head>
    <title>HTML5 Input Types</title>
</head>
<body>
    <h1>HTML5 Adds New Input Types</h1>

    Date: <input type="date"> <br/> <br/>
    Month: <input type="month"> <br/> <br/>
    Week: <input type="week"> <br/> <br/>
    Time: <input type="time"> <br/> <br/>
    Date/Time: <input type="datetime"> <br/> <br/>

</body>
</html>
```

CSS could be improved.

But which would your users prefer?

Fallback to a library.

Speaking of pickers.

Color!

Pick a color, get a hex value!

Cool!

But it's only in Firefox, Opera and Chrome.

Bummer.

Uses native picker.

Several new attributes.

Most of them are boolean: if present, it is on.

Autofocus.

Autocomplete - on or off.

Partial support across the browsers right now.

Placeholder.

Hint displayed in an input field.

Several form related attributes.

formaction, formenctype, formmethod, formnovalidate

Mostly has to do with submitting forms.

Validation!

Validation is on by default.

Simplest is required.

Also works with some of the new input types.

Email address.

URL.

Numbers.

Even respects min/max.

Don’t want to validate?

Use novalidate attribute.

Support is...improving.

Of course you will still validate on the server…

Lab time!

Forms

- Take the sample web page and "convert" it to use HTML5 form elements
- Make a field required
- Put focus in the first field
- Add placeholder text
- Submit the form in various browsers
- ${extract}/html5_workshop/labs/forms.html

Fun With Numbers

- Convert the text field to a number
- Write a function that adds 1 to the number
- Write a function that subtracts 1 from the number
- Write a function to display the type of the input field using valueAsNumber
- ${extract}/html5_workshop/labs/numbers.html

Canvas.

Graphics!

Graphs, shapes, animations, etc.

Controlled via scripting.

Pretty simple.

```
<canvas id="canvas" width="800" height="800"></canvas>
```

That's it?

Only two attributes: *width* and *height*.

Optional, default is
300 pixels by 150 pixels.

CSS sizing as well.

Can be styled like an image -
boarder, margin, etc.

Specifying fallback content.

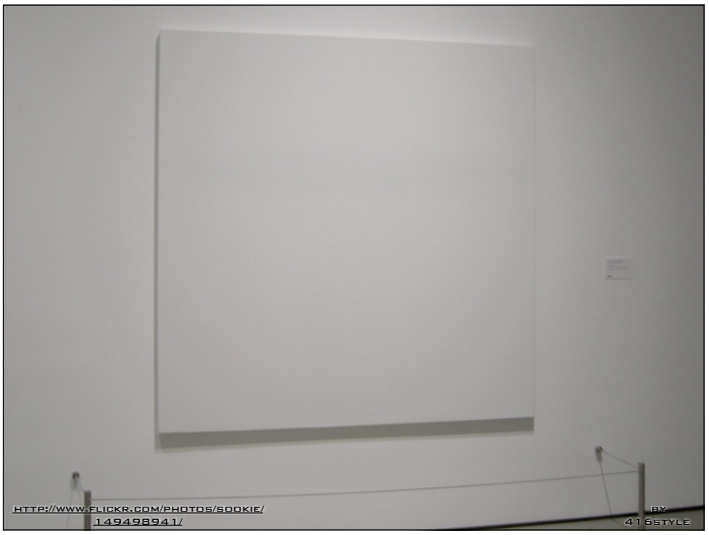
```
<canvas id="fallback" width="800" height="800">
   Put fallback content here...perhaps an image.
</canvas>
```

Content between tag.

Ignored by browsers supporting canvas...

Canvas tag is ignored by browsers lacking support.

Canvas starts like any canvas...



Up to you to fill it!

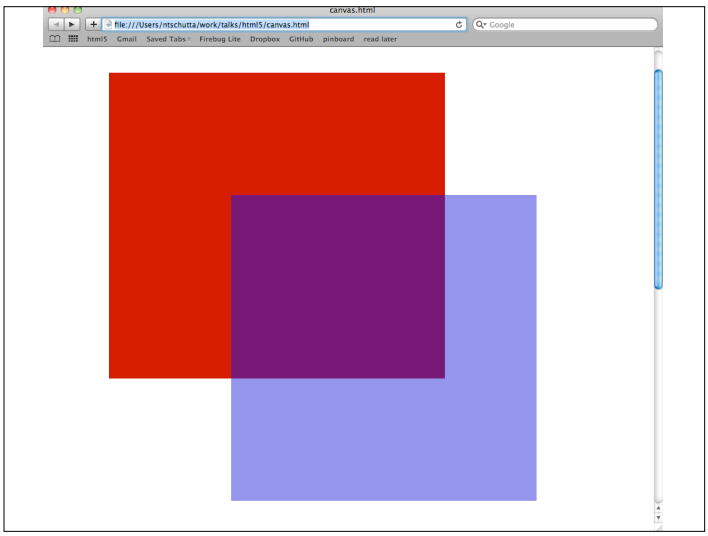
To draw, we need a context.

```
var ctx = canvas.getContext("2d");
```

For now, just 2D...

Likely 3D in the future.

Once we have a context, we can draw!




```
<head>
    <script type="application/javascript">
        function draw() {
            var canvas = document.getElementById("canvas");
            var ctx = canvas.getContext("2d");

            ctx.fillStyle = "rgb(200,0,0)";
            ctx.fillRect(100, 100, 550, 500);

            ctx.fillStyle = "rgba(0, 0, 200, 0.5)";
            ctx.fillRect(300, 300, 550, 500);
        }
    </script>
</head>
<body onload="draw()">
<canvas id="canvas" width="800" height="800"></canvas>

<canvas id="fallback" width="800" height="800">
   Put fallback content here...perhaps an image.
</canvas>
```

One primitive - rectangle.

Three methods.

```
fillRect(x, y, width, height);
strokeRect(x, y, width, height);
clearRect(x, y, width, height);
```

All take the same arguments...

x and y position of left corner of rectangle...

Width and height.

fillRect - filled rectangle.

strokeRect - outline.

clearRect - clears area,
makes it transparent.

So that's it? Rectangles?

No!

Paths.

We can draw shapes.

- *beginPath* - creates path
- *closePath* - tries to close the shape
- *stroke* - draws an outlined shape
- *fill* - draws a solid shape
- *moveTo* - moves the "pen", doesn't draw

https://developer.mozilla.org/en/Canvas_tutorial%3aDrawing_shapes

```
function draw() {
    var canvas = document.getElementById("canvas");
    var ctx = canvas.getContext("2d");

    ctx.beginPath();
    ctx.arc(75,75,50,0,Math.PI*2,true); // Outer circle
    ctx.moveTo(110,75);
    ctx.arc(75,75,35,0,Math.PI,false);   // Mouth (clockwise)
    ctx.moveTo(65,65);
    ctx.arc(60,65,5,0,Math.PI*2,true);  // Left eye
    ctx.moveTo(95,65);
    ctx.arc(90,65,5,0,Math.PI*2,true);  // Right eye
    ctx.stroke();

}
```

Arc? Draws circles.

arc(x, y, radius, startAngle, endAngle, anticlockwise)

lineTo(x, y) - Straight lines

Curves.

quadraticCurveTo,
bezierCurveTo

And of course you can combine these...

You can also use images.

Get an image - from page of from scratch.

drawImage(image, x, y)

Useful as backdrops...

Can also scale images.

Add width and height.

You can also crop...

And on and on!

Colors, gradients, line styles, patterns...

Rotating, scaling, transforms, compositing.

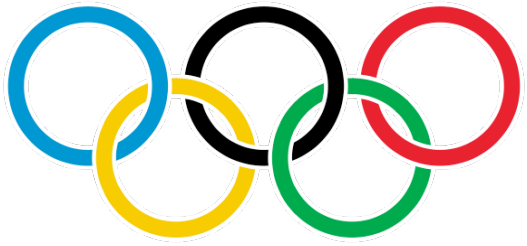
Animations!

Whew!

Lab time!

Olympic Rings

- Using the various canvas methods, draw the Olympic Rings
- ${extract}/html5_workshop/labs/canvas_rings.html



That's not all!

You can create text too.

Fairly low level API.

Give it the text to write,
the font, weight, color, etc.

```
context.font
context.textalign
```

bold 1.5em sans-serif

Stuff like that.

But hey, it's CSS.

`context.fillText(text, x, y)`

Also `strokeText`.

Lab time!

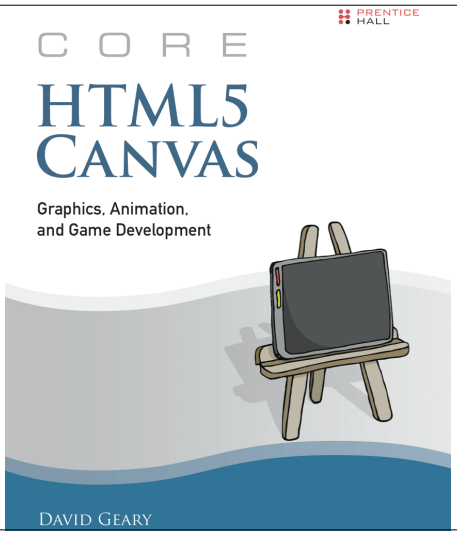
Writing on a Canvas

- Using the various canvas text methods:
  - Draw the text from an input box
  - Change the color of the text based on user input
  - Allow the user to specify italics
- ${extract}/html5_workshop/labs/canvas_text.html

Again, very rich space.

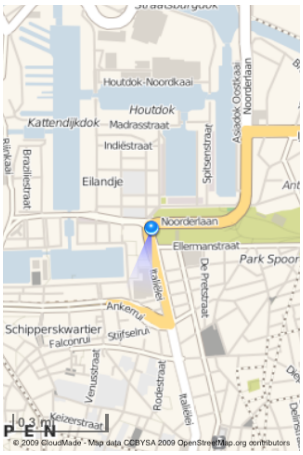
Want more?

http://corehtml5canvas.com



Geolocation.

Where in the world are you?



http://www.offmaps.com/

Very helpful on phones!

Technically not a part of HTML5.

Geolocation Working Group.

Wide browser support.

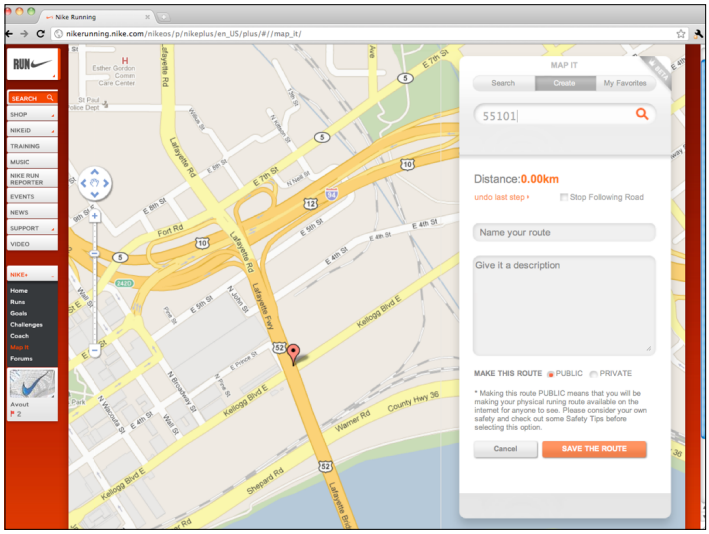
Your browser doesn't support it?

Device specific options.

Privacy issue?

Absolutely.

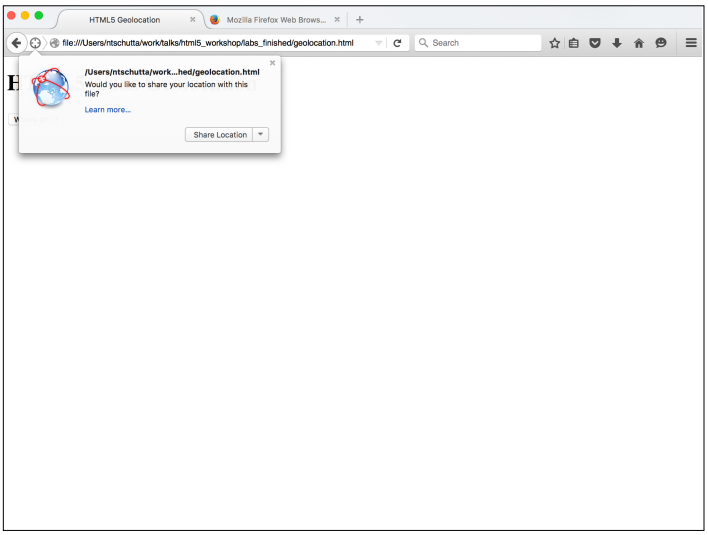
If you know where the device is...



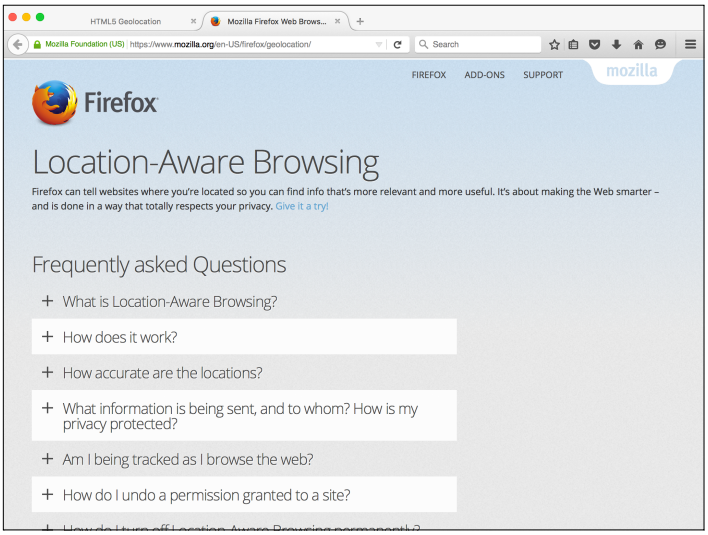
Opt-in.

http://www.w3.org/TR/geolocation-API/#security

Browsers tell you
before data is sent.



Some provide more info.



Infobars are smart.

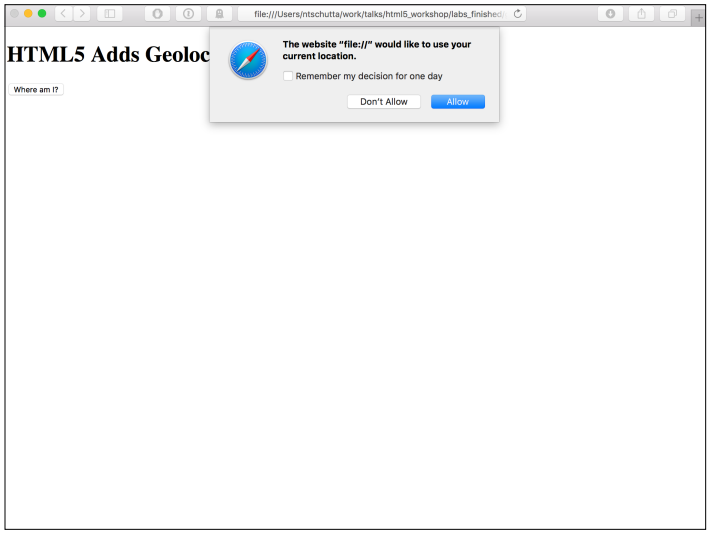
Often give link to
further information.

Aren't modal.

Tab specific.

Blocks.

How does this work?




```
function whereAmI() {
  if (Modernizr.geolocation) {
    navigator.geolocation.getCurrentPosition(showLocation);
  } else {
    alert("this browser doesn't support geolocation, sorry!");
  }
}

function showLocation(position) {
  var latitude = position.coords.latitude;
  var longitude = position.coords.longitude;
  $("#location").html("lat: " + latitude + " and long: " + longitude);
}
```

New property.

navigator.geolocation

Simple API.

getCurrentPosition()

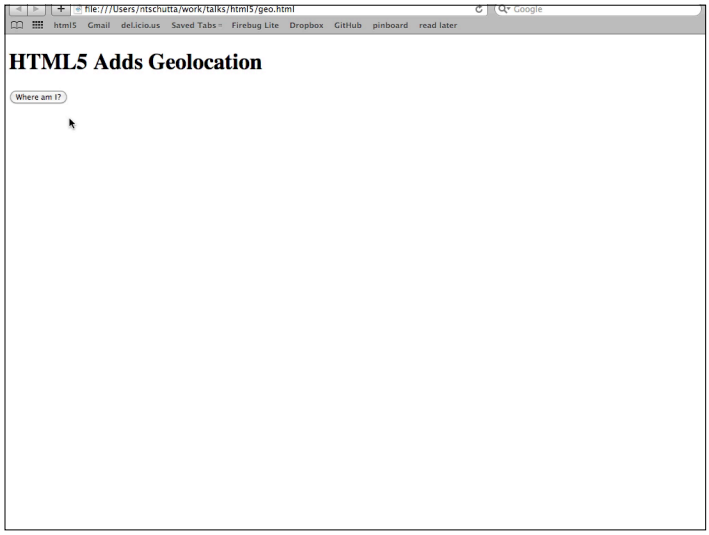
Browser "determines location," creates `Position`.

`Position` contains `Coordinates` & timestamp.

Coordinates

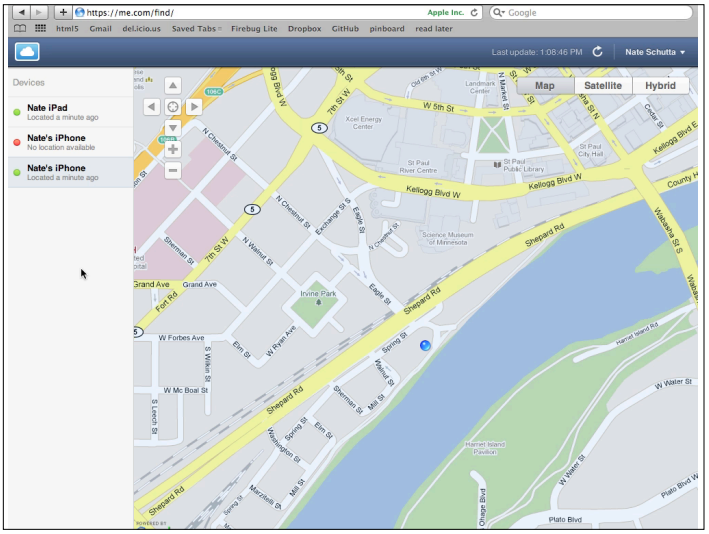
- latitude (double)
- longitude (double)
- altitude (double or `null`)
- accuracy (double)
- altitudeAccuracy (double or `null`)
- heading (double or `null`)
- speed (double or `null`)

You supply a callback function.



How accurate is it?

Can be *very* accurate.



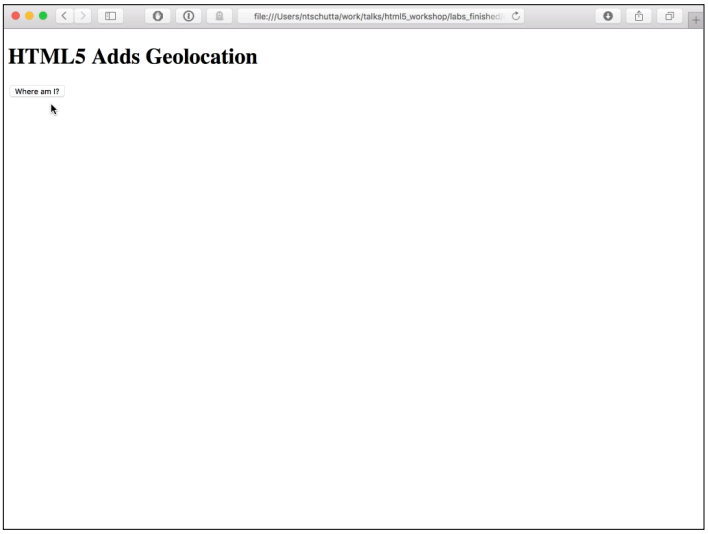
Your function receives a Position object with:

coords & timestamp

Can take some time ;)

getCurrentPosition()

Optional second arg: error callback function.



```
function whereAmI() {
  if (Modernizr.geolocation) {
    navigator.geolocation.getCurrentPosition(showLocation, handleError);
  } else {
    alert("this browser doesn't support geolocation, sorry!");
  }
}

function showLocation(position) {
  var latitude = position.coords.latitude;
  var longitude = position.coords.longitude;
  $("#location").html("lat: " + latitude + " and long: " + longitude);
}

function handleError(error) {
  if(error.code == 1) {
    var message = "You want to keep your location private, that's OK!"
  }
  $("#location").html("Ooops! " + message);
}
```

Callback gets a `PositionError` object.

Two attributes.

code & message

Code values...

- PERMISSION_DENIED (1)
- POSITION_UNAVAILABLE (2)
- TIMEOUT (3)
- UNKNOWN_ERROR (0)

Don't have to use the magic numbers.

error.PERMISSION_DENIED
error.POSITION_UNAVAILABLE
error.TIMEOUT
error.UNKNOWN_ERROR

getCurrentPosition()

Optional third argument.

PositionOptions

- enableHighAccuracy (boolean)
- timeout (long)
- maximumAge (long)

All attributes are optional.

Higher accuracy
may be slower.

Some devices have separate permissions.

Timeout is based on network time, not user.

Age allows you to cache positions.

`watchPosition()`

Generally need an accurate GPS device, like a phone.

Callback when the position data changes.

When the user moves or you get more accurate data.

Returns an ID.

`clearWatch(id)` stops the monitoring of location.

IE? Out of luck < 9.

Are other options.

Gears, device specific.

geo.js

http://code.google.com/p/geo-location-javascript/

Layer over various approaches.

Lab time!

Geolocation

- Using the geolocation API, create a function that displays your current latitude and longitude
- Handle the situation where a user opts out
- Handle the situation where location cannot be determined
- ${extract}/html5_workshop/labs/geolocation.html

Local storage.

Technically web storage.

Split into separate spec.

Some browsers call it DOM Storage.

Simple way to store *key/value* pairs.

Like cookies...

But bigger, stays local.

Persistent.

No expiration date.

Close the browser?
Data is still there.

Storage is per domain.

Very wide support.

Key is a *string*.

Data can be any
JavaScript datatype...

But it's stored as a *string*.

Don't forget to parse...

Simple interface.

getItem(key)
setItem(key, value)

setItem() *silently overwrites* old values.

getItem() with unused key *returns null*...

Can treat localStorage as an associative array.

In other words, bracket notation.

localStorage.getItem("key")
localStorage["key"]

Can remove items:
removeItem(key)

Called on a nonexistent key does nothing.

localStorage.length gets number of stored values.

key(index) retrieves key at that index.

Index out of bounds returns null.

Also a storage event.

Calls to setItem, removeItem or clear...

Provided something changes.

StorageEvent

- key
- newValue
- oldValue
- storageArea
- url

Can't cancel it.

Default max:
5 megabytes.

Exceed that...
QUOTA_EXCEEDED_ERR

Can you ask for more?

No.

Some browsers allow the user to control quota.

Note there is also session storage.

Data lives for on one session.

Close the window? Data is deleted.

Lab time!

Local Storage

- Add a function that stores the current values locally
- Add a function that retrieves the values in local storage
- Add a function that displays the locally stored values
- Add a function that clears local storage
- ${extract}/html5_workshop/labs/localstorage_form.html

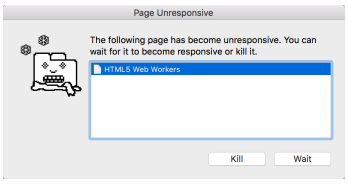
Web Workers.

JavaScript - single threaded.

Makes things easy.

Optimized DOM access.

But now we see the dreaded slow scripts.



And we do a lot in JavaScript today!

That's a problem.

Want a responsive UI.

How do we do that?

Run scripts in the background.

Independent of the UI.

Not interrupted by user actions.

User continues on...

Do what we need to do behind the scenes.

Relatively heavy weight.

Can hog resources!

Be careful...

How does it work?

First, no direct access to the DOM.

Message passing.

How do we do it?

```
var foo = new Worker("foo.js");
```

Create a new worker.

foo.js contains the worker code.

Pass message to worker.

```
foo.postMessage(input);
```

What can we send?

String, array, JSON object...

Cannot send a function.

How do we get messages?

Define a callback function.

foo.onmessage

Get an event object.

event.data = message from the worker.

event.target = which worker sent the message.

In the worker, define onmessage.

Allows it to
receive messages.

```
onmessage = callFib;
```

Function handles
messages sent to worker.

Worker responds with
`postMessage();`

Messages between page
and worker are copied.

That's it!

Easy enough?

Couple more things...

`foo.terminate();`

Removes the worker.

Running scripts abort.

Can call `close()` from the worker itself.

foo.onerror

Error handler.

Workers can create workers!

Same technique.

Just from the original worker.

Lab time!

Web Workers

- Implement calculateFibWW
- Convert fibWW.js to be a web worker
- Use a web worker to call fibWW.js
- ${extract}/html5_workshop/labs/web_worker.html

Selectors.

We've all written this...

document.getElementById

Ever fat finger that?

jQuery taught us a better way...

jQuery selectors.

CSS 1-3.

Plus more ;)

Very powerful.

How did we live without it for so long?

HTML5 borrows from jQuery selectors.

document.querySelector

Simple syntax.

Powerful selectors.

Can be very precise.

By id, class, tag, child elements, combinations, etc.

What about multiple matches?

Should return the first one.

What if you need a collection of elements?

document.querySelectorAll

Returns an array of elements that match the selector.

Very useful.

Lab time!

jQuery Style Selectors

- Display the student's name retrieving the element by ID
- Display the student's age retrieving the element by class
- Display the first item in the list retrieving the element with a child selector
- Display the third item in the list retrieving all the li elements
- ${extract}/html5_workshop/labs/selectors.html

Drag and Drop.

Many modern web apps include drag and drop.

Various libraries, numerous implementations.

Yet another nod to what we're doing!

Specify something as draggable...

Say where it can be dropped...

Supply handlers for the various events.

What can be draggable?

Almost anything.

draggable="true"

Easy to customize.

dataTransfer property - send data.

Events.

- dragstart
- dragend
- dragenter
- dragleave
- dragover
- drop

Very flexible.

Allows for some advanced techniques.

Drag items between windows...

Dragging files!

Desktop to browser...
or vice versa.

More focussed on
data transfer.

Very useful!

Libraries will eventually defer
to native implementations.

Lab time!

HTML5 Drag and Drop

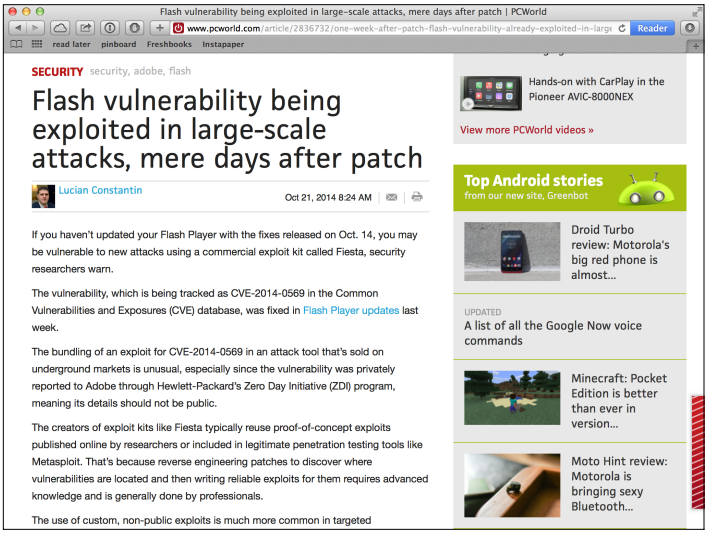
- Make the two images draggable
- Make the text div draggable
- Make the ul draggable
- When dropped on the target, add the dragged item to the target
- ${extract}/html5_workshop/labs/dragndrop.html

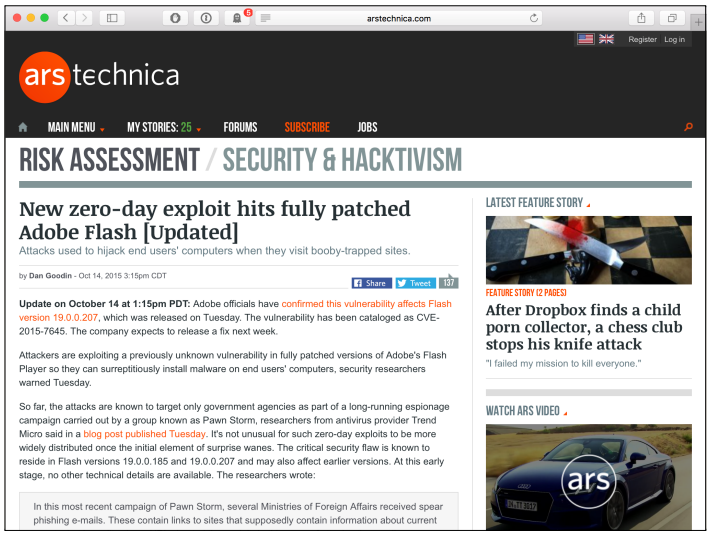
Video.

Video on the web isn't new.

But it's always involved a plugin.

And those plugins haven't always been...all that safe.





It'd be nice if the browser could just play a video.

Now it can!

Now we have a video element.

Give it a height and width.

Source for the video.

Looks a lot like the image tag.

Codecs are still an issue.

Likely need to encode in more than one.

Sorry.

Though Firefox now supports H.264.

http://arstechnica.com/gadgets/2012/03/idealism-vs-pragmatism-mozilla-debates-supporting-h264-video-playback/

Lab time!

Video

- Using the video element, show the video found in the images folder
- ${extract}/html5_workshop/labs/video.html

History.

Forward/back buttons.

Back in the day...

Remove the chrome!

Hope users don't use keyboard shortcuts.

Why do we need back buttons?

The browser has one.

History API.

window.history

Want to change the URL?

history.pushState()

Browser won't load the URL.

Doesn't have to exist.

Three parameters.

1. State object.

JavaScript object.

Anything that's serializable.

Object associated with the new history element.

2. Title.

Typically ignored.

3. URL.

Can be relative.

Must be from the same origin.

Protocol, domain, port must remain the same.

Page doesn't load.

Creates an entry in browser history.

Backwards navigation throws `popstate` event.

Your app can respond accordingly.

`history.replaceState()`

Works like pushState...

Modifies current history entry.

Support is decent.

History.js.

https://github.com/balupton/history.js

Lab time!

History API

- Add "dummy" pages to the browser's history stack
- Implement the forward and back buttons to navigate the history stack
- Implement the length function to show how many items are in history
- ${extract}/html5_workshop/labs/history.html

Offline.

Google Apps, GMail rock.

Some flights have wifi.

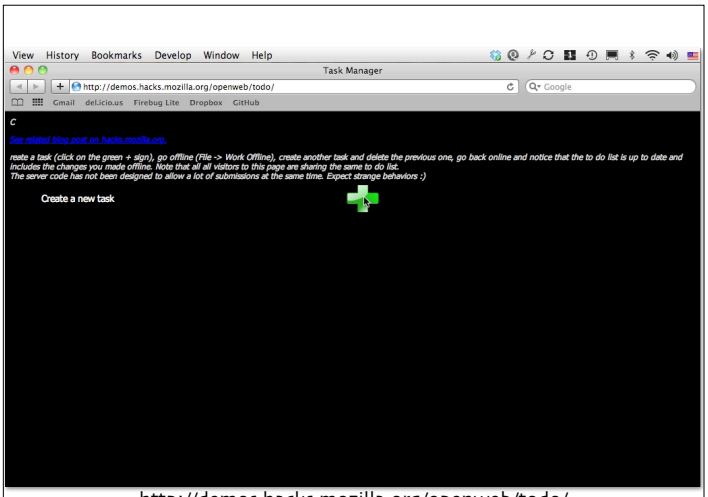
Not cheap.

Application cache.

Web server tells client what it needs.

Application works when disconnected.

When it connects, changes are uploaded.



http://demos.hacks.mozilla.org/openweb/todo/

New events: *online* and *offline*.

Hard part?

What to do when you're back online...

Lab time!

Offline Detector

- Create a method that shows the network status
- Bind to the online and offline events triggering the method
- ${extract}/html5_workshop/labs/offline.html

Sort of...

Hacks.

Comet.

aka Ajax Push, HTTP Streaming, Reverse Ajax...

Long polling.

Applets.

Flash.

CometD.

Dojo Foundation.

Bayeux Protocol.

All of these methods have various issues.

Some proved difficult for developers.

Others involved complex streaming protocols.

Many stumbled on proxies and firewalls.

Impacted servers.

Latency and throughput.

HTTP 1.1 spec itself...

Limited browser connections!

We could do it...but
it wasn't easy.

HTML5 is huge.

But you don't have to do it all to embrace it.

Pick and choose.

What scratches your itch?

What solves your pain.

Adopt that.

Specs are evolving.

So are the browsers.

Keep a weather eye out.

Questions?!?

Thanks!

Nathaniel T. Schutta
@ntschutta