Modernize gUM

Why we shouldn’t wait

Reading prep for this presentation

JS Arrow functions is used for briefer examples:

```
navigator.getUserMedia(constraints, function(stream) { }, function(reason) { });
```

```
navigator.getUserMedia(constraints, stream => { }, reason => { });
```

They’re still callback functions.

Reading prep for this presentation

Promises - why we are here.

```
foo(result => log("foo's callback called w/" + result), failure);
```

```
foo().then(result => log("foo fulfilled w/" + result)).catch(failure);
```

Powerful standardized pattern/error-handling/prose

http://stackoverflow.com/questions/22539815/arent-promises-just-callbacks

WebIDL-supported: http://heycam.github.io/webidl/#idl-promise

Standard prose: https://w3ctag.github.io/promises-guide

Spec: http://people.mozilla.org/~jorendorff/es6-draft.html

Paradigm shift has already happened

Anything new in the last year with async values uses promises. WebRTC is main callback holdout.

- https://w3c.github.io/screen-orientation
- https://w3c.github.io/push-api
- HTML's createImageBitmap()
- https://slightlyoff.github.io/ServiceWorker/spec/service_worker

Why? Big reason: Error-handling

Lets reexamine the code from two slides ago:

what if log throws?

, pass in failure?

```
foo(result => log("foo's callback called w/" + result), failure);
```

or try/catch around log?

```
foo().then(result => log("foo fulfilled w/" + result)).catch(failure);
```

if log throws it's caught by this final catch and sent to failure

Compare to our error-handling

```
navigator.getUserMedia(stream => {
  doSomething(stream);
  stream.getTracks().forEach(track => pc.addTrack(track, stream));
  pc.createOffer({ }, step2, failure);
}, failure);
```

If doSomething throws, progress ends, but it's uncaught!

failure never hears of it. STALL!!

There's the same disconnect between synchronous errors (exceptions) and asynchronous errors. Both halt progress, but the former is uncaught and only the latter is propagated out.

Problem: Our error-handling is broken

Error-handling in our current API is fraught with peril and should not be exposed to anyone. Promises fix this.

Plain callbacks without some exception-plan are not OK for async APIs. Where do exceptions go? Spec doesn't say. Users must vigilantly try/catch everything:

```
var failure = reason => log("Failed to show camera: " + reason.message);
navigator.getUserMedia(stream => { try {
    videoElement.mozSrcObject = stream;
    videoElement.play();
  } catch(e) { failure(e); } //failure must tolerate non-MediaStreamError
}, failure);
```

Problem (continued)

MediaStreamError is incompatible with Promises.

Our API guarantees that error-callbacks get a MediaStreamError, which rules out propagation of other errors. Also, MediaStreamError doesn't inherit from Error, which is against the promise spec recommendation:

[1] says "Promise rejection reasons should always be instances of the ECMAScript Error type, just like synchronously-thrown exceptions should always be instances of Error as well.

In particular, for DOM or other web platform specs, this means you should never use DOMError, but instead use DOMException, which per WebIDL extends Error."

[1] https://w3ctag.github.io/promises-guide/#reasons-should-be-errors

What's proposed - getUserMedia

```
navigator.mediaDevices.getUserMedia(constraints).then(stream => {
  videoElement.mozSrcObject = stream;
  videoElement.play();
}).catch(reason => log("Failed to show camera: " + reason.message));

// Discouraged legacy getUserMedia with callbacks intact on navigator
navigator.getUserMedia(constraints, stream => {
  videoElement.mozSrcObject = stream;
  videoElement.play();
}, reason => log("Failed to get camera: " + reason.message));
```

https://github.com/w3c/mediacapture-main/pull/18

What's proposed - applyConstraints

```
videoTrack.applyConstraints({ frameRate: { min: 60 } })
  .then(() => log("Yeah!"), reason => log("Nope! " + reason.message))
  .catch(reason => log("Proper errhandling rulz! " + reason.message));
```

Not yet implemented in Firefox; others?

https://github.com/w3c/mediacapture-main/pull/18

What's proposed - enumerateDevices

```
navigator.mediaDevices.enumerateDevices().then(devices =>
  devices.forEach(device => log("Device: " + device.label));
}).catch(reason => log("Couldn't display devices: " + reason.message));
```

Introduced June 16th. Not yet implemented in Firefox; others?

Better name may be: `navigator.mediaDevices.enumerate()`

since one can only say "device" so many times!

https://github.com/w3c/mediacapture-main/pull/18

Why? Standardised prose

Before:

When enumerateDevices() is called, the UA must queue a task that runs the following steps:

1. Let *resultCallback* be the first argument.
2. Let *resultList* be an empty list.
3. ..
4. Invoke *resultCallback* with *resultList* as its argument.

Parameters:

resultCallback `MediaDeviceInfoCallback`

Return: `void`

After:

When enumerateDevices() is called, the UA must run the following steps:

1. Let *p* be a new promise.
2. Run the following steps in parallel:
    1. Let *resultList* be an empty list.
    2. ...
    3. Resolve *p* with *resultList*.
3. Return *p*.

No parameters.

Return:

`Promise<sequence<MediaDeviceInfo>>`

Why? WebRTC is next! <3

```
function dial(pc, signal) {
  return mediaDevices.getUserMedia(constraints)
  .then(stream => {
    stream.getTracks().forEach(track =>
      pc.addTrack(track, stream));
  })
  .then(()=> pc.createOffer(options))
  .then(offer => pc.setLocalDescription(offer))
  .then(() => signal.then(answer =>
    pc.setRemoteDescription(answer)));
}
```

```
function pickup(pc, signal) {
  return mediaDevices.getUserMedia(constraints)
  .then(stream => {
    stream.getTracks().forEach(track =>
      pc.addTrack(track, stream));
  })
  .then(() => signal.then(offer =>
    pc.setRemoteDescription(offer)))
  .then(() => pc.createAnswer(options))
  .then(answer => pc.setLocalDescription(answer));
}
```

```
Promise.all([dial(pc1, pc2.stable), pickup(pc2, pc1.haveLocalOffer)])
    .then(() => log("Connected!"))
    .catch(failed); // 1-line error handling
```

Degrees of backwards compatibility

1. In Specification

   Number of existing uses so widespread across multiple implementations that all future browsers must implement it as truth for all eternity (navigator.getUserMedia)

2. Implementation-specific

   Existing use limited to early adopters (who will be quick to adapt again!)
   Browser continues its support transitionally with web-console warnings. Other browsers not required to adopt. Spec is clean (webkitMediaDevices.enumerateDevices?)

3. Behold to the webkit/moz-prefix Action Card!

   Browser breaks early adopters because prefixes!
   (navigator.getUserMedia({ facingMode: “user”, require[“facingMode”] }, success, fail))

The webkit/moz-prefix Action Card

Technical value: Bupkis

Psychological value: “Get out of jail free”!

Discard after play.

Lets player say “Told you!” and break smaller things. All early adopters must fix their stuff to comply for the benefit of the group, and like it.

Backwards-comp. polyfills (if needed)

```
function getUserMedia(constraints, success, failure) {
  var p = navigator.mediaDevices.getUserMedia(constraints);
  p.then(success);
  p.then(failure);
}

function enumerateDevices(success) {
  var p = navigator.mediaDevices.enumerate();
  p.then(success);
}
```

Summary: Reasons to do this now

- Fix our broken error-handling.
- Modern specification language.
- Inevitability. In a year's time we'll regret not having done this (proof: a year ago).
- Fan-out. WebRTC is next. Whatever we decide here likely impacts what WebRTC decides.
- Decent backwards compatibility.
- Last chance while we're still prefixed!