Modernize WebRTC *

* Just the basics

1. ~~**Why**~~
2. **What's proposed**
3. **Compatibility**

What's proposed - Offer/Answer

If it took callbacks, it now returns a promise instead

```
pc1.createOffer(options)
.then(offer => pc1.setLocalDescription(offer))
.then(() => pc2.setRemoteDescription(pc1.localDescription))
.then(() => pc2.createAnswer())
.then(answer => pc2.setLocalDescription(answer))
.then(() => pc1.setRemoteDescription(pc2.localDescription))
.then(() => log("Connected!"))
.catch(reason => log("Failed to connect! " + reason.message));

// A complete local-loop offer/answer-exchange in 8 lines of code!
```

https://github.com/w3c/webrtc-pc/pull/11

Compatibility approach: Hybrid

```
var pc = RTCPeerConnection(); // promise and callbacks

pc.createAnswer(success, fail);                 // OK
var promise = pc.createAnswer();                // OK
var promise = pc.createAnswer(success, fail); // OK
```

Trivial to implement. Even the lone difficult one:

```
pc.createOffer(success, fail, options);         // OK
var promise = pc.createOffer(options);          // OK
var promise = pc.createOffer(optionsOrSuccess, fail, options); // OK
```

https://github.com/w3c/webrtc-pc/pull/12

Clean Hybrid: WebIDL overloading

```
interface RTCPeerConnection : EventTarget  {

  Promise<RTSessionDescription> createOffer (optional RTCOfferOptions options);

  void createOffer (RTCSessionDescriptionCallback successCallback,
                    RTCPeerConnectionErrorCallback failureCallback,
                    optional RTCOfferOptions options);
  …
};
```

https://github.com/w3c/webrtc-pc/pull/??

Summary

- Remove our broken error-handling.
- Simpler and safer to use. Better API. Modern.
- Synergy with MediaCapture and ImageCapture.
- WebRTC = super-asynchronous = super-benefits.
- Inevitability. In a year’s time we’ll regret not having done this (proof: a year ago).

Appendix

(rest of presentation)

Reading prep for this presentation

JS Arrow functions are used for briefer examples:

```
pc.createOffer(function(stream) { }, function(reason) { }, options);
```

```
pc.createOffer(stream => { }, reason => { }, options);
```

They’re still callback functions.

Reading prep for this presentation

Promise - unified primitive for asynchronicity.

```
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

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

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

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.

- http://w3c.github.io/mediacapture-main
- http://gmandyam.github.io/image-capture
- http://w3c.github.io/screen-orientation
- http://w3c.github.io/push-api
- http://html.spec.whatwg.org#dom-createimagebitmap
- http://slightlyoff.github.io/ServiceWorker/spec/service_worker

Big reason: Our error-handling is broken

```
pc.createAnswer(answer => {
  try {
    doSomething(answer);
    pc.setLocalDescription(answer);
  } catch(e) { failure(e); }
}, failure);
```

what if doSomething throws? Things stop, so say why!

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 is not OK for async APIs. Where do exceptions go? Spec doesn't say. Users must vigilantly try/catch everything.

Promise use-case #2: concurrency

```
var alice = new RTCPeerConnection(), bob = new RTCPeerConnection();
Promise.all([dial(alice, bob.stable), pickup(bob, alice.haveLocalOffer)])
.then(() => log("Connected!"))
.catch(failed); // 1-line error handling in this faster local-loop call-setup
```

```
function dial(pc, signal) {
  return mediaDevices.getUserMedia(constraints)
  .then(stream => pc.addStream(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 => pc.addStream(stream))
  .then(() => signal.then(offer =>
    pc.setRemoteDescription(offer)))
  .then(() => pc.createAnswer(options))
  .then(answer => pc.setLocalDescription(answer));
}
```

Compatibility approach #1: Prefixes

```
var pc = RTCPeerConnection();       // promise
var pc = webkitRTCPeerConnection(); // callbacks
var pc = mozRTCPeerConnection();    // callbacks
```

```
var PeerConnection = window.RTCPeerConnection || // BAD! breaks!
                     window.webkitRTCPeerConnection ||
                     window.mozRTCPeerConnection;
```

```
var PeerConnection = window.webkitRTCPeerConnection ||
                     window.mozRTCPeerConnection ||
                     window.RTCPeerConnection; // GOOD!
```

Upside: fixing is trivial + compatible.

Backwards-compatible pseudo polyfill

```
// for setLocalDescription, setRemoteDescription & addIceCandidate
var _foo = RTCPeerConnection.prototype.foo;
RTCPeerConnection.prototype.foo = function(arg, success, fail) {
  return (success || fail)? _foo(arg).then(success, fail) : _foo(arg);
}

// for icky createOffer (createAnswer is trivial)
var _bar = RTCPeerConnection.prototype.bar;
RTCPeerConnection.prototype.bar = function(argOrSuccess, fail, arg) {
  return (!argOrSuccess || isArg(argOrSuccess))?
     _bar(argOrSuccess) : _bar(arg).then(success, fail);
}
```