Peer-to-peer Data API

A few slides to get the discussion going

Example

```
function start(offer) {
    pc = new PeerConnection(null);

    // send any ice candidates to the other peer
    pc.onicecandidate = function (evt) {
        signalingChannel.send(JSON.stringify({ "type": "candidate", "sdp": evt.candidate }));
    };

    var type;
    if (!offer) {
        // create data channel and setup chat
        channel = pc.createDataChannel();
        setupChat(channel);

        pc.createOffer(gotDescription);
        type = "offer";
    } else {
        // setup chat on incoming data channel
        pc.datachannel = function (evt) {
            setupChat(evt.channel);
        };

        pc.setRemoteDescription("offer", offer);
        pc.createAnswer(offer, gotDescription);
        type = "answer";
    }

    function gotDescription(desc) {
        pc.setLocalDescription(type, desc);
        signalingChannel.send(JSON.stringify({ "type": type, "sdp": desc }));
    }
}

function sendChatMessage(msg) {
    channel.send(msg);
}

function setupChat(channel) {
    channel.onopen = function () {
        enableChat(channel);
    };

    channel.onmessage = function (evt) {
        newChatMessage(evt.data);
    };
}

signalingChannel.onmessage = function (evt) {
    var msg = JSON.parse(evt.data);
    switch (msg.type) {
    case "offer":
        // create the PeerConnection
        start(new SessionDescription(msg.sdp));
        break;
    case "answer":
        pc.setRemoteDescription(msg.type, new SessionDescription(msg.sdp));
        break;
    case "candidate":
        pc.addIceCandidate(new IceCandidate(msg.sdp));
        break;
    }
};
```

Create PeerConnection and handle generated candidates

Initiator creates channel and offers

Receiver waits for incoming channel, processes offer, and creates answer

Manage chat (enable chat, send/receive messages)

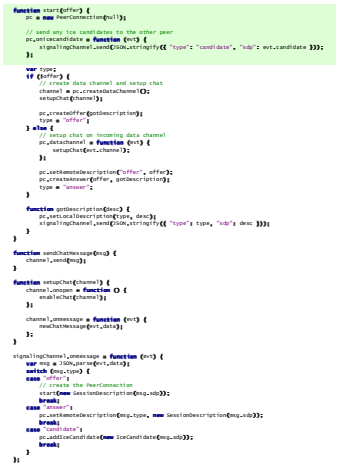
Process incoming offers, answers, and candidates

Example Walkthrough

```
function start(isCaller) {
    pc = new PeerConnection(null);

    // send any ice candidates to the other peer
    pc.onicecandidate = function (evt) {
        signalingChannel.send(
                JSON.stringify({ "type": "candidate", "sdp": evt.candidate }));
    };

    ...
```



Example Walkthrough

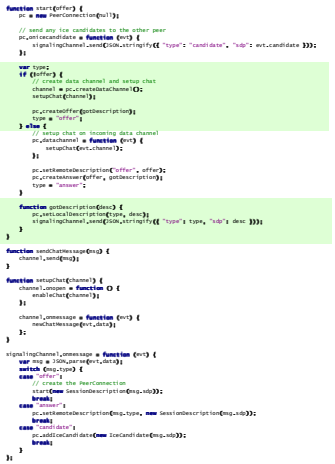
```
    ...

    var type;
    if (!offer) {
        // create data channel and setup chat
        channel = pc.createDataChannel();
        setupChat(channel);

        pc.createOffer(gotDescription);
        type = "offer";
    } else {

    ...

    function gotDescription(desc) {
        pc.setLocalDescription(type, desc);
        signalingChannel.send(JSON.stringify(
                { "type": type, "sdp": desc }));
    }
}
```



Example Walkthrough

```
    ...

    } else {
        // setup chat on incoming data channel
        pc.datachannel = function (evt) {
            setupChat(evt.channel);
        };

        pc.setRemoteDescription("offer", offer);
        pc.createAnswer(offer, gotDescription);
        type = "answer";
    }

    function gotDescription(desc) {
        pc.setLocalDescription(type, desc);
        signalingChannel.send(JSON.stringify(
                { "type": type, "sdp": desc }));
    }
}
```

Example Walkthrough

```
function sendChatMessage(msg) {
    channel.send(msg);
}

function setupChat(channel) {
    channel.onopen = function () {
        enableChat(channel);
    };

    channel.onmessage = function (evt) {
        newChatMessage(evt.data);
    };
}
```

Example Walkthrough

```
signalingChannel.onmessage = function (evt) {
    var msg = JSON.parse(evt.data);
    switch (msg.type) {
    case "offer":
        // create the PeerConnection
        start(new SessionDescription(msg.sdp));
        break;
    case "answer":
        pc.setRemoteDescription(msg.type, new SessionDescription(msg.sdp));
        break;
    case "candidate":
        pc.addIceCandidate(new IceCandidate(msg.sdp));
        break;
    }
```

```
interface DataChannel {
   readonly attribute DOMString      label;
   readonly attribute boolean        reliable;
   const unsigned short CONNECTING = 0;
   const unsigned short OPEN = 1;
   const unsigned short CLOSING = 2;
   const unsigned short CLOSED = 3;
   readonly attribute unsigned short readyState;
   readonly attribute unsigned long  bufferedAmount;
   [TreatNonCallableAsNull]
         attribute Function?      onopen;
   [TreatNonCallableAsNull]
         attribute Function?      onerror;
   [TreatNonCallableAsNull]
         attribute Function?      onclose;
   void close ();
   [TreatNonCallableAsNull]
         attribute Function?      onmessage;
         attribute DOMString      binaryType;
   void send (DOMString data);
   void send (ArrayBuffer data);
   void send (Blob data);
};

interface PeerConnection {
    ...
    DataChannel createDataChannel ([TreatNullAs=EmptyString] DOMString? label,
                                                              optional DataChannelInit? dataChannelDict);
              attribute Function?          ondatachannel;
    ...
};


[Constructor(DOMString type, DataChannelEventInit eventInitDict)]
interface DataChannelEvent : Event {
    readonly attribute DataChannel channel;
};
dictionary DataChannelEventInit : EventInit {
    DataChannel channel;
};
```