

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 }));  
  };  
  
  ...  
}
```

```
function startOffer() {  
  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("chat");  
    setupChat(channel);  
    pc.ondatachannel = function (evt) {  
      type = "offer";  
    };  
  } else {  
    // setup chat on incoming data channel  
    pc.ondatachannel = function (evt) {  
      setupChat(evt.channel);  
    };  
  }  
  
  pc.onicecandidate = function (evt) {  
    if (evt.candidate) {  
      pc.onicecandidate = function (evt) {  
        type = "answer";  
      };  
    }  
  };  
  
  function getLocalDescription() {  
    return pc.getLocalDescription();  
  }  
  signalingChannel.send(JSON.stringify({ "type": "type", "data": desc }));  
}  
  
function sendAnswerOffer() {  
  channel.send(desc);  
}  
  
function attachToChannel() {  
  channel.addEventListener("message", function (e) {  
    chat.receive(e.data);  
  });  
  channel.addEventListener("close", function (e) {  
    chat.receive(e.data);  
  });  
  channel.addEventListener("error", function (e) {  
    chat.receive(e.data);  
  });  
}  
  
signalingChannel.onmessage = function (evt) {  
  var msg = JSON.parse(evt.data);  
  switch (msg.type) {  
    case "offer":  
      // create the PeerConnection  
      startOffer();  
      break;  
    case "answer":  
      pc.onicecandidate = function (evt) {  
        type = "offer";  
      };  
      break;  
    case "candidate":  
      pc.onicecandidate = function (evt) {  
        type = "candidate";  
      };  
      break;  
  }  
}
```

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 }));
}
}
```

```
function startOffer() {
  pc = new PeerConnection(null);
  // send any ice candidates to the other peer
  pc.onIceCandidate = function (event) {
    if (signalingChannel.readyState === "open") {
      signalingChannel.send(JSON.stringify({ type: "candidate", "sdp": event.candidate.sdp }));
    }
  };

  var type;
  // create data channel and setup chat
  channel = pc.createDataChannel("chat");
  setupChat(channel);
  pc.createOffer(gotDescription);
  type = "offer";
} else {
  // setup chat on incoming data channel
  pc.onDataChannel = function (event) {
    pc.createDataChannel("offer", {offerTo: pc.remoteSessionId});
    type = "answer";
  };
}

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

function sendAnswer(desc) {
  channel.send(desc);
}

function attachToChannel() {
  channel.addEventListener("open", function () {
    channel.send("connected");
  });
  channel.addEventListener("close", function () {
    channel.send("disconnected");
  });
}

signalingChannel.onmessage = function (event) {
  var msg = JSON.parse(event.data);
  switch (msg.type) {
    case "offer":
      // create the PeerConnection
      case "candidate":
        break;
      case "type":
        pc.createOffer(msg.type, new PeerConnectionOptions({iceServers: []}));
        break;
      case "candidate":
        pc.addIceCandidate(new RTCIceCandidate(msg.candidate));
        break;
    }
  }
}
```

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 }));  
}
```

```
function startOffer() {  
  pc = new PeerConnection(null);  
  // send any ice candidates to the other peer  
  onIceCandidate = function (evt) {  
    if (signalingChannel.send(JSON.stringify({ type: "candidate", sdp: evt.candidate })))  
    };  
  
  var type;  
  if (offer) {  
    // create data channel and setup chat  
    channel = pc.createDataChannel("C");  
    setupChat(channel);  
    pc.setLocalDescription(offer);  
    type = "offer";  
  } else {  
    // setup chat on incoming data channel  
    pc.datachannel = function (evt) {  
      setupChat(evt.channel);  
    };  
    pc.setRemoteDescription(offer);  
    pc.createAnswer(offer, offer, gotDescription);  
    type = "answer";  
  }  
  
  function gotDescription(desc) {  
    pc.setLocalDescription(type, desc);  
    signalingChannel.send(JSON.stringify({ type: type, sdp: desc }));  
  }  
  
  function sendAnswer(desc) {  
    channel.send(desc);  
  }  
  
  function attachToChannel() {  
    channel.onopen = function () {  
      attachToChannel();  
    };  
    channel.onmessage = function (evt) {  
      handleMessage(evt.data);  
    };  
  }  
  
  signalingChannel.onmessage = function (evt) {  
    var msg = JSON.parse(evt.data);  
    switch (msg.type) {  
      case "offer":  
        // create the PeerConnection  
        startOffer();  
        break;  
      case "answer":  
        pc.setRemoteDescription(msg.type, new TextDescription(msg.sdp));  
        break;  
      case "candidate":  
        pc.addIceCandidate(new TextCandidate(msg.sdp));  
        break;  
    }  
  }  
}
```

Example Walkthrough

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

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

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

```
function startEvt() {  
    pc = new PeerConnection({});  
    // send my ice candidates to the other peer  
    onIceCandidate = function (evt) {  
        if (!evt.candidate) {  
            signalingChannel.send(JSON.stringify({ type: "candidate", "id": myCandidate }));  
        }  
    };  
    var types = ["audio", "video"];  
    // create data channel and setup chat  
    channel = pc.createDataChannel("chat");  
    setupChat(channel);  
    pc.setLocalDescription();  
    type = "offer";  
} else {  
    // setup chat on incoming data channel  
    onDataChannel = function (evt) {  
        setupChat(channel);  
    };  
    pc.onDataChannel = onDataChannel; // offer or  
    pc.onIceCandidate = offerOrCandidate; // offer or  
    type = "answer";  
} function getLocalDescription() {  
    pc.getLocalDescription();  
    signalingChannel.send(JSON.stringify({ type: "type", "id": desc }));  
}  
}  
  
function sendChatMessage(msg) {  
    channel.send(msg);  
}  
  
function enableChat(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  
            pc = new PeerConnection({});  
            break;  
        case "answer":  
            pc.setLocalDescription(msg.type, msg.getLocalDescription());  
            break;  
        case "candidate":  
            pc.addIceCandidate(msg.candidate);  
            break;  
    }  
}
```

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;  
  }  
}
```

```
function startOffer() {  
  pc = new PeerConnection(null);  
  // send any ice candidates to the other peer  
  onIceCandidate = function (evt) {  
    signalingChannel.send(JSON.stringify({ type: "candidate", "id": evt.candidate.id }));  
  };  
  var type;  
  if (offer) {  
    // create data channel and setup chat  
    channel = pc.createDataChannel("chat", {  
      negotiated: true,  
      ordered: true,  
      binaryType: "blob" });  
    pc.ondatachannel = function (event) {  
      pc.attachChannel(channel);  
    };  
    // setup chat on incoming data channel  
    pc.onmessage = function (evt) {  
      signalingChannel.send(JSON.stringify({ type: "text", "data": evt.data }));  
    };  
  }  
  pc.setLocalDescription("offer", offer, onIceCandidate);  
  pc.onicecandidate = function (evt) {  
    if (evt.candidate) {  
      signalingChannel.send(JSON.stringify({ type: "candidate", "id": evt.candidate.id }));  
    }  
  };  
  pc.onnegotiationneeded = function () {  
    signalingChannel.send(JSON.stringify({ type: "text", "data": "negotiation needed" }));  
  };  
  signalingChannel.send(JSON.stringify({ type: "offer", "sdp": offer.sdp }));  
}
```

```

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;
};

```