# WebRTC Video Chat in 20 Lines of JavaScript (Part 1)

* By [Kevin Gleason](https://www.pubnub.com/blog/author/keving/)

* Follow [@](http://twitter.com/)

* August 25, 2015

**Since we wrote this post, we’ve made some changes to our what we do, and what we don’t do with WebRTC.**[Learn more here.](https://www.pubnub.com/developers/demos/webrtc/)

WebRTC, so hot right now. If you haven’t heard of it, WebRTC (Web Realtime Communications) is an API that enables peer-to-peer video, audio, and data communication in a web browser with no plugins, frameworks, or applications required.

In this tutorial, we’ll start by building a very simple video chat application that only requires around 20 lines of JavaScript. Our application will enable users to enter a username and call another user between browsers. [In our next part, we’ll add some cool features to bolster our WebRTC video app.](https://www.pubnub.com/blog/2015-08-25-webrtc-group-video-chatting-basics/)

Check out the live [WebRTC video chat demo here](http://kevingleason.me/SimpleRTC/minivid.html), open up two windows, and watch it in action!

[And if you want to check out the project code for the tutorial series, it’s available here.](https://github.com/pubnub/SimpleRTC)

**And the**[project code for this specific project here](https://github.com/pubnub/SimpleRTC/blob/gh-pages/minivid.html)**.**

That’s right! Let’s get to it.



This image represents two web browsers.

### Quick Note on Testing and Debugging

If you try to open file://<your-webrtc-project> in your browser, you will likely run into Cross-Origin Resource Sharing (CORS) errors since the browser will block your requests to use video and microphone features.

To test your code you have a few options. You can upload your files to a web server, like [Github Pages](https://pages.github.com/) if you prefer. However, to keep development local, I recommend you setup a simple server using Python.

To do this, open your terminal and change directories into your current project and depending on your version of Python, run one of the following modules.

|  |  |
| --- | --- |
|  | cd <project-dir> |
|  |  |
|  | # Python 2 |
|  | python -m SimpleHTTPServer <portNo> |
|  |  |
|  | # Python 3 |
|  | python -m http.server <portNo> |

[**view raw**](https://gist.github.com/GleasonK/209de622507e433fa69d/raw/8734aa9127357ce86b61f298eaef8d210a3227af/bash.sh)[**bash.sh**](https://gist.github.com/GleasonK/209de622507e433fa69d#file-bash-sh) hosted with ❤ by [**GitHub**](https://github.com/)

For example, I run Python2.7 and the command I use is python -m SimpleHTTPServer 8001. Now I can go to http://localhost:8001/index.html to debug my app. Try making an index.html with anything in it and serve it on localhost before you continue.

### Step 1: The HTML5 Backbone

For the sake of the demo, let’s keep the HTML short and simple. First we need a div to house our videos. Then, all we really need to start off with is a login field so you can specify your name and a call field so you can dial someone.

|  |  |
| --- | --- |
|  | <div id="vid-box"></div> |
|  |  |
|  | <form name="loginForm" id="login" action="#" onsubmit="return login(this);"> |
|  | <input type="text" name="username" id="username" placeholder="Pick a username!" /> |
|  | <input type="submit" name="login\_submit" value="Log In"> |
|  | </form> |
|  |  |
|  | <form name="callForm" id="call" action="#" onsubmit="return makeCall(this);"> |
|  | <input type="text" name="number" placeholder="Enter user to dial!" /> |
|  | <input type="submit" value="Call"/> |
|  | </form> |

[**view raw**](https://gist.github.com/GleasonK/209de622507e433fa69d/raw/8734aa9127357ce86b61f298eaef8d210a3227af/01.html)[**01.html**](https://gist.github.com/GleasonK/209de622507e433fa69d#file-01-html) hosted with ❤ by [**GitHub**](https://github.com/)

This should leave you with an elaborate, well styled HTML file that looks something like this:

[minivid_html](https://www.pubnub.com/wp-content/uploads/2015/08/minivid_html.png)

### Step 2: The JavaScript Imports

There are three libraries that you will need to include to make WebRTC operations much easier:

1. Include [jQuery](https://jquery.com/) to make modifying DOM elements a breeze.
2. Include the [PubNub JavaScript SDK](https://www.pubnub.com/docs/web-javascript/pubnub-javascript-sdk) to facilitate the WebRTC signaling (more on signaling at the bottom of this blog post).
3. Include the [PubNub WebRTC SDK](https://github.com/pubnub/webrtc) to make placing phone calls as simple as calling the dial(number) function.

|  |  |
| --- | --- |
|  | <script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script> |
|  | <script src="https://cdn.pubnub.com/pubnub-3.7.14.min.js"></script> |
|  | <script src="https://cdn.pubnub.com/webrtc/webrtc.js"></script> |

[**view raw**](https://gist.github.com/GleasonK/0557fdd279e95349c896/raw/ae0288255a9a27442f9f3833655d02593d4fe169/02.html)[**02.html**](https://gist.github.com/GleasonK/0557fdd279e95349c896#file-02-html) hosted with ❤ by [**GitHub**](https://github.com/)

Now we’re ready to write our calling functions for login and makeCall.

### Step 3: Preparing to Receive Calls

In order to start facilitating video calls, you will need a publish and subscribe key. To get your pub/sub keys, you’ll first need to [sign up for a PubNub account](https://admin.pubnub.com/#/register?abtest=&psc=WC-Tracking&pt=wc-tracking&PN_First_Page_Visited=https%3A%2F%2Fwww.pubnub.com%2Fblog%2F2015-08-25-webrtc-video-chat-app-in-20-lines-of-javascript%2F&PN_Page_Before_Sign_Up=https://www.pubnub.com/blog/2015-08-25-webrtc-video-chat-app-in-20-lines-of-javascript/). Once you sign up, you can find your unique PubNub keys in the [PubNub Developer Dashboard](https://admin.pubnub.com/). The free Sandbox tier should give you all the bandwidth you need to build and test your WebRTC application.

First, lets use JavaScript to find our video holder, where other callers faces will go.

|  |  |
| --- | --- |
|  | var video\_out = document.getElementById("vid-box"); |

[**view raw**](https://gist.github.com/GleasonK/209de622507e433fa69d/raw/8734aa9127357ce86b61f298eaef8d210a3227af/03.js)[**03.js**](https://gist.github.com/GleasonK/209de622507e433fa69d#file-03-js) hosted with ❤ by [**GitHub**](https://github.com/)

Next, we’ll implement the login function. This function will set up the phone using the username they provided as a UUID.

|  |  |
| --- | --- |
|  | function login(form) { |
|  | var phone = window.phone = PHONE({ |
|  | number : form.username.value || "Anonymous", // listen on username line else Anonymous |
|  | publish\_key : 'your\_pub\_key', |
|  | subscribe\_key : 'your\_sub\_key', |
|  | }); |
|  | phone.ready(function(){ form.username.style.background="#55ff5b"; }); |
|  | phone.receive(function(session){ |
|  | session.connected(function(session) { video\_out.appendChild(session.video); }); |
|  | session.ended(function(session) { video\_out.innerHTML=''; }); |
|  | }); |
|  | return false; // So the form does not submit. |
|  | } |

[**view raw**](https://gist.github.com/GleasonK/209de622507e433fa69d/raw/8734aa9127357ce86b61f298eaef8d210a3227af/04.js)[**04.js**](https://gist.github.com/GleasonK/209de622507e433fa69d#file-04-js) hosted with ❤ by [**GitHub**](https://github.com/)

You can see we use the username as the phone’s number, and instantiate PubNub using your own publish and subscribe keys. The next function, phone.ready, allows you to define a callback for when the phone is ready to place a call. I simply change the username input’s background to green, but you can tailor this to your needs.

The phone.receive function allows you to define a callback that takes a session as a parameter for when a call event occurs, whether that be a new call, a call hangup, or for losing service, you attach those event handlers to the sessions in phone.receive.

I defined session.connected which is invoked after receiving a phone call, and when you are ready to begin video chatting. I simply appended the session’s stream to our video div.

Then, I define session.ended which is called after invoking phone.hangup. This is where you place end-call logic. I simply clear the video holder’s innerHTML.

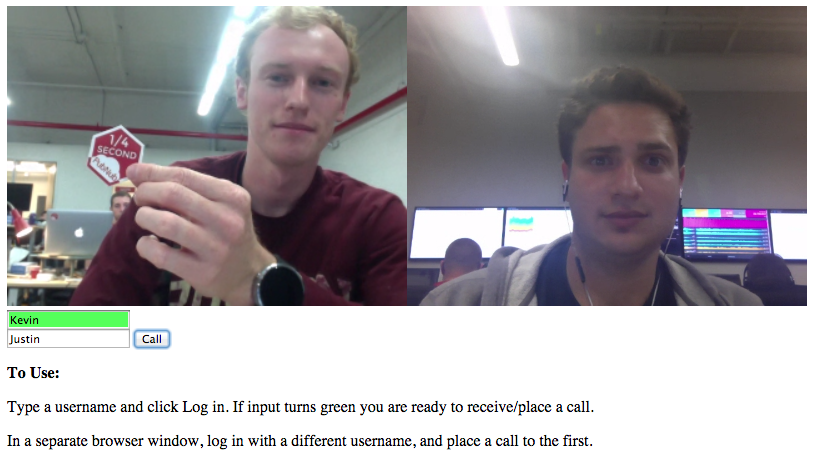
### Step 4: Making Calls

We now have a our WebRTC video app ready to receive a call, so it is time to create a makeCall function.

|  |  |
| --- | --- |
|  | function makeCall(form){ |
|  | if (!window.phone) alert("Login First!"); |
|  | else phone.dial(form.number.value); |
|  | return false; |
|  | } |

[**view raw**](https://gist.github.com/GleasonK/209de622507e433fa69d/raw/8734aa9127357ce86b61f298eaef8d210a3227af/05.js)[**05.js**](https://gist.github.com/GleasonK/209de622507e433fa69d#file-05-js) hosted with ❤ by [**GitHub**](https://github.com/)

If window.phone is undefined, we cannot place a call. This will happen if the user did not log in first. If it exists, we use the phone.dial function which takes a number and an optional list of servers to place a call.

[](https://www.pubnub.com/wp-content/uploads/2015/08/chat.png)

And that is it! You now have a simple WebRTC video chat app. Fire up your python server and go test your app on localhost!

In our next two parts, we walkthrough how to add a number of additional features to your WebRTC video chat application, including: make/end Calls, thumbnail streams, mute call, pause video, and group chatting. We’ll also walk through how to create live embeddable streaming content (ie. how to build Periscope with WebRTC!).

## Why PubNub? Signaling.

WebRTC is not a standalone API, it needs a signaling service to coordinate communication. Metadata needs to be sent between callers before a connection can be established.

This metadata includes things such as:

* Session control messages to open and close connections
* Error messages
* Codecs/Codec settings, bandwidth and media types
* Keys to establish a secure connection
* Network data such as host IP and port

Once signaling has taken place, video/audio/data is streamed directly between clients, using WebRTC’s PeerConnection API. This peer-to-peer direct connection allows you to stream high-bandwidth robust data, like video.

PubNub makes this signaling incredibly simple, and then gives you the power to do so much more with your WebRTC applications.

### Browser Compatibility

WebRTC is widely adopted by popular browsers such as Chrome and Firefox, but there are many browsers on which certain features will not work. See a list of [supported browsers here.](http://iswebrtcreadyyet.com/)

### Want to learn more?

Good, that never-ending quest for knowledge will get you far in life. Here are some other resources PubNub offers on WebRTC:

* [PubNub WebRTC SDK](https://github.com/stephenlb/webrtc-sdk)
* [What is WebRTC](https://www.pubnub.com/blog/2014-01-14-what-is-webrtc/)?
* [PubNub WebRTC Video Chat Demo](https://www.pubnub.com/developers/demos/webrtc/)

[Now, time for Part Two! In our next part, we’ll show you how to add a number of useful WebRTC features to your chat app.](https://www.pubnub.com/blog/2015-08-25-webrtc-group-video-chatting-basics/)

Read More: [JavaScript](https://www.pubnub.com/blog/tag/javascript/), [VOIP or WebRTC Signaling](https://www.pubnub.com/blog/tag/voip-or-webrtc-signaling/), [WebRTC](https://www.pubnub.com/blog/tag/webrtc/)

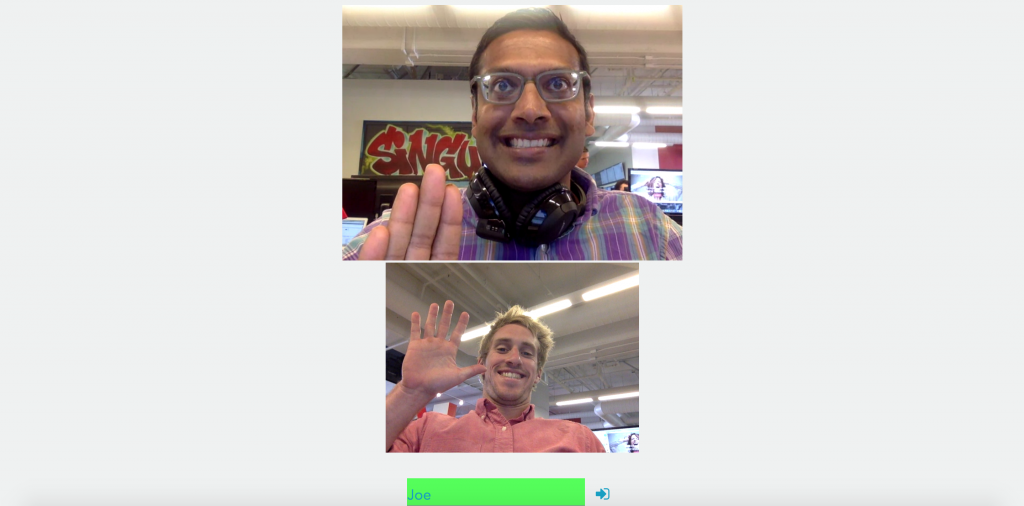
WebRTC Group Video Chatting Basics (Part 2)

* By [Kevin Gleason](https://www.pubnub.com/blog/author/keving/)

* Follow [@](http://twitter.com/)

* August 25, 2015

Since we wrote this post, we’ve made some changes to our what we do, and what we don’t do with WebRTC. [Learn more here.](https://www.pubnub.com/developers/demos/webrtc/)

[](https://www.pubnub.com/wp-content/uploads/2015/08/Screen-Shot-2015-08-24-at-12.40.01-PM.png)

In this part, we’ve added a feature that allows the caller to see themselves, as well as the person they’re chatting with.

Picking up where we left off from [Part One: WebRTC Video Chat in 20 Lines of JavaScript](https://www.pubnub.com/blog/2015-08-25-webrtc-video-chat-app-in-20-lines-of-javascript/), it’s time to bolster our WebRTC video chat application. If you haven’t seen [Part One](https://www.pubnub.com/blog/2015-08-25-webrtc-video-chat-app-in-20-lines-of-javascript/) yet, check it out then come back here!

In this tutorial, we’ll show you how to use a wrapper class to provide some basic group chat functionality to your WebRTC video chat app. Building off our simple WebRTC chat app we built in [Part One](https://www.pubnub.com/blog/2015-08-25-webrtc-video-chat-app-in-20-lines-of-javascript/), we’ll add the following features:

* Make/End Calls
* Thumbnail Streams (the caller can see themselves, as well as the person they’re chatting with)
* Mute Call
* Pause Video
* Group Chatting

Check out the live [WebRTC video chat demo here](http://kevingleason.me/SimpleRTC/minivid2.html), open up two windows, and watch it in action!

[And if you want to check out the project code for the tutorial series, it’s available here.](https://github.com/pubnub/SimpleRTC)

A Note on Testing and Debugging

If you try to open file://<your-webrtc-project> in your browser, you will likely run into Cross-Origin Resource Sharing (CORS) errors since the browser will block your requests to use video and microphone features. To test your code you have a few options. You can upload your files to a web server, like [Github Pages](https://pages.github.com/) if you prefer. However, to keep development local, I recommend you setup a simple server using Python.

To do this, open your terminal and change directories into your current project and depending on your version of Python, run one of the following modules.

|  |  |
| --- | --- |
|  | cd <project-dir> |
|  |  |
|  | # Python 2 |
|  | python -m SimpleHTTPServer <portNo> |
|  |  |
|  | # Python 3 |
|  | python -m http.server <portNo> |

[**view raw**](https://gist.github.com/GleasonK/c4968791dab659a05804/raw/1f40cf26666a588e6379faa62f86df3e4da7a8de/bash.sh)[**bash.sh**](https://gist.github.com/GleasonK/c4968791dab659a05804#file-bash-sh) hosted with ❤ by [**GitHub**](https://github.com/)

For example, I run Python2.7 and the command I use is python -m SimpleHTTPServer 8001. Now I can go to http://localhost:8001/index.html to debug my app! Try making an index.html with anything in it and serve it on localhost before you continue.

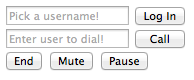
Step 1: The HTML5 Backbone

For the sake of the demo, let’s keep the HTML short and simple. First we need a div to house our videos. Then, we will make login field so you can specify your name, a call field so you can dial someone, and a few buttons to implement features.

|  |  |
| --- | --- |
|  | <div id="vid-box"></div> |
|  | <div id="vid-thumb"></div> |
|  |  |
|  | <form name="loginForm" id="login" action="#" onsubmit="return login(this);"> |
|  | <input type="text" name="username" id="username" placeholder="Pick a username!" /> |
|  | <input type="submit" name="login\_submit" value="Log In"> |
|  | </form> |
|  |  |
|  |  |
|  | <form name="callForm" id="call" action="#" onsubmit="return makeCall(this);"> |
|  | <input type="text" name="number" placeholder="Enter user to dial!" /> |
|  | <input type="submit" value="Call"/> |
|  | </form> |
|  |  |
|  | <div id="inCall"> <!-- Buttons for in call features --> |
|  | <button id="end" onclick="end()">End</button> <button id="mute" onclick="mute()">Mute</button> <button id="pause" onclick="pause()">Pause</button> |
|  | </div> |

[**view raw**](https://gist.github.com/GleasonK/c4968791dab659a05804/raw/1f40cf26666a588e6379faa62f86df3e4da7a8de/01.html)[**01.html**](https://gist.github.com/GleasonK/c4968791dab659a05804#file-01-html) hosted with ❤ by [**GitHub**](https://github.com/)

This should leave you with an elaborate, well styled HTML file that looks something like this:

[](https://www.pubnub.com/wp-content/uploads/2015/08/minivid2_html.png)

You will notice there are a few functions that we will need to create in the JavaScript:

* – login(form): Prepare the phone to receive calls and setup functionality.
* – makeCall(form): Make a call to the number in the form.
* – end(): Hangup the current call.
* – mute(): Mute your local audio stream.
* – pause(): Pause your video stream from sending.

Step 2: The JavaScript Imports

There are three libraries that you will need to include to make WebRTC operations much easier:

1. Include [jQuery](https://jquery.com/) to make modifying DOM elements a breeze.
2. Include the [PubNub JavaScript SDK](https://www.pubnub.com/docs/web-javascript/pubnub-javascript-sdk) to facilitate the WebRTC signaling.
3. Include the [PubNub WebRTC SDK](https://github.com/pubnub/webrtc), and SDK Wrapper libraries which makes placing phone calls as simple as calling the dial(number) function.

|  |  |
| --- | --- |
|  | <script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script> |
|  | <script src="https://cdn.pubnub.com/pubnub-3.7.14.min.js"></script> |
|  | <script src="https://cdn.pubnub.com/webrtc/webrtc.js"></script> |
|  | <script src="https://cdn.pubnub.com/webrtc/rtc-controller.js"></script> |

[**view raw**](https://gist.github.com/GleasonK/0557fdd279e95349c896/raw/ae0288255a9a27442f9f3833655d02593d4fe169/2-02.html)[**2-02.html**](https://gist.github.com/GleasonK/0557fdd279e95349c896#file-2-02-html) hosted with ❤ by [**GitHub**](https://github.com/)

Now we are ready to write our calling functions for login and makeCall!

Step 3: Receiving Calls

In order to start facilitating video calls, you will need a publish and subscribe key. To get your pub/sub keys, you’ll first need to [sign up for a PubNub account](https://admin.pubnub.com/#/register?pi_visitorid=235674065&abtest=&psc=WC-Tracking&pt=wc-tracking&PN_First_Page_Visited=https%3A%2F%2Fwww.pubnub.com%2Fblog%2F2015-08-25-webrtc-video-chat-app-in-20-lines-of-javascript%2F&PN_Page_Before_Sign_Up=https://www.pubnub.com/blog/2015-08-25-webrtc-group-video-chatting-basics/). Once you sign up, you can find your unique PubNub keys in the [PubNub Developer Dashboard](https://admin.pubnub.com/). The free Sandbox tier should give you all the bandwidth you need to build and test your WebRTC Application.

3.1 Prepare to Receive

First, let’s locate our video holder, where other callers’ faces go, and thumbnail holder, where our video stream will be held.

|  |  |
| --- | --- |
|  | var video\_out = document.getElementById("vid-box"); |
|  | var vid\_thumb = document.getElementById("vid-thumb"); |

[**view raw**](https://gist.github.com/GleasonK/c4968791dab659a05804/raw/1f40cf26666a588e6379faa62f86df3e4da7a8de/03.js)[**03.js**](https://gist.github.com/GleasonK/c4968791dab659a05804#file-03-js) hosted with ❤ by [**GitHub**](https://github.com/)

Now, to implement the login function. This function will set up the phone using the username provided in the form as a UUID.

|  |  |
| --- | --- |
|  | function login(form) { |
|  | var phone = window.phone = PHONE({ |
|  | number : form.username.value || "Anonymous", // listen on username line else Anonymous |
|  | publish\_key : 'your\_pub\_key', |
|  | subscribe\_key : 'your\_sub\_key', |
|  | }); |
|  | var ctrl = window.ctrl = CONTROLLER(phone); |
|  | ctrl.ready(function(){ ... }); // Called when ready to receive call |
|  | ctrl.receive(function(session){ |
|  | session.connected(function(session){ ... }); // New Call |
|  | session.ended(function(session) { ... }); // Call Ended |
|  | }); // Called on incoming call/call ended |
|  | ... |
|  | return false; //prevents form from submitting |
|  | } |

[**view raw**](https://gist.github.com/GleasonK/c4968791dab659a05804/raw/1f40cf26666a588e6379faa62f86df3e4da7a8de/04.js)[**04.js**](https://gist.github.com/GleasonK/c4968791dab659a05804#file-04-js) hosted with ❤ by [**GitHub**](https://github.com/)

This is the basis of our login function. The whole purpose of login is to set up our phone and attach all the features we want to it.

PHONE({configs}) is a constructor from the PubNub WebRTC SDK, while CONTROLLER(phone) is the wrapper library that attaches many useful functions to your phone object. PHONE receives a config object with pub/sub keys and a phone number.

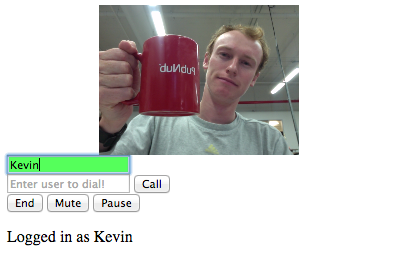
The ctrl wrapper also does some behind the scenes user management, so call ctrl.ready and ctrl.receiveinstead of their phone counterparts.

Now, let’s look at those functions. ctrl.ready is called when you have authorized video and mic permissions and your phone is ready to receive calls. This means our local stream is ready and we can add it to our thumbnail div using ctrl.addLocalStream.

|  |  |
| --- | --- |
|  | ctrl.ready(function(){ |
|  | form.username.style.background="#55ff5b"; // Turn input green |
|  | form.login\_submit.hidden="true"; // Hide login button |
|  | ctrl.addLocalStream(vid\_thumb); // Place local stream in div |
|  | }); |

[**view raw**](https://gist.github.com/GleasonK/c4968791dab659a05804/raw/1f40cf26666a588e6379faa62f86df3e4da7a8de/05.js)[**05.js**](https://gist.github.com/GleasonK/c4968791dab659a05804#file-05-js) hosted with ❤ by [**GitHub**](https://github.com/)

Now when you test your app, you should see your local stream in the thumbnail holder.

[](https://www.pubnub.com/wp-content/uploads/2015/08/minivid2_thumb.png)

3.2 Handle Incoming Calls

Time to define ctrl.receive, which is called any time there is a new incoming call, or a user ends a call. This callback is handed a session, variable which holds all the data about the joining/leaving user, including their video stream.

The simplest thing you can do here is appendChild to your video\_out div, and remove it when they exit.

|  |  |
| --- | --- |
|  | ctrl.receive(function(session){ |
|  | session.connected(function(session){ video\_out.appendChild(session.video); }); |
|  | session.ended(function(session) { ctrl.getVideoElement(session.number).remove(); }); |
|  | }); |

[**view raw**](https://gist.github.com/GleasonK/c4968791dab659a05804/raw/1f40cf26666a588e6379faa62f86df3e4da7a8de/06.js)[**06.js**](https://gist.github.com/GleasonK/c4968791dab659a05804#file-06-js) hosted with ❤ by [**GitHub**](https://github.com/)

All it takes to add a video stream to your window is video\_out.appendChild(session.video). However, a controller function ctrl.getVideoElement(number) is used to get the users video element from your screen, then removes it.

All user’s video elements have a HTML5 data tag with their number on it, use this to your advantage. All the getVideoElement function does is locate a video element by that data tag using $('\*[data-number="'+number+'"]').

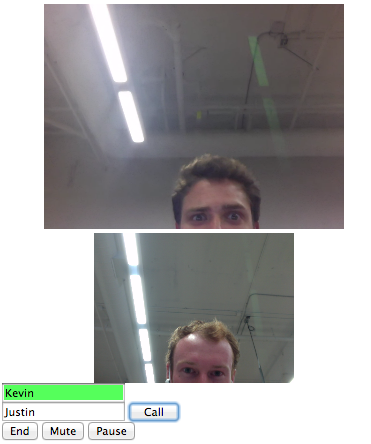
Step 4: Making Calls

We now have a phone ready to receive a call, so it is time to create a makeCall function.

|  |  |
| --- | --- |
|  | ctrl.receive(function(session){ |
|  | session.connected(function(session){ video\_out.appendChild(session.video); }); |
|  | session.ended(function(session) { ctrl.getVideoElement(session.number).remove(); }); |
|  | }); |

[**view raw**](https://gist.github.com/GleasonK/c4968791dab659a05804/raw/1f40cf26666a588e6379faa62f86df3e4da7a8de/06.js)[**06.js**](https://gist.github.com/GleasonK/c4968791dab659a05804#file-06-js) hosted with ❤ by [**GitHub**](https://github.com/)

If window.phone is undefined, we cannot place a call. This will hafppen if the user did not log in first. If it is, we then check to see if a user is online with ctrl.isOnline(number, function(isOn){...} This will check if num is online. Their status will be passed to the second parameter, a callback function. In that callback function we simply use an if statement. If the user is online, ctrl.dial(number) to dial them and begin a video chat.

[](https://www.pubnub.com/wp-content/uploads/2015/08/minivid2_chat.png)

Testing time! Fire up your python server and go test your app on localhost. When you are finished staring at yourself in the thumbnail, you may proceed.

Step 5: Additional Features

We still have a few functions to implement, namely end, mute, and pause. Hanging up a call should be easy, so we made it easy.

|  |  |
| --- | --- |
|  | function end(){ |
|  | ctrl.hangup(); |
|  | } |

[**view raw**](https://gist.github.com/GleasonK/c4968791dab659a05804/raw/1f40cf26666a588e6379faa62f86df3e4da7a8de/08.js)[**08.js**](https://gist.github.com/GleasonK/c4968791dab659a05804#file-08-js) hosted with ❤ by [**GitHub**](https://github.com/)

Now, muting a call and pausing a video stream offers some nice customization. The controller wrapper has functions ctrl.toggleVideo() and ctrl.toggleAudio(). Both return a boolean whether video and audio is enabled after the toggle.

|  |  |
| --- | --- |
|  | function mute(){ |
|  | var audio = ctrl.toggleAudio(); |
|  | if (!audio) $("#mute").html("Unmute"); |
|  | else $("#mute").html("Mute"); |
|  | } |
|  |  |
|  | function pause(){ |
|  | var video = ctrl.toggleVideo(); |
|  | if (!video) $('#pause').html('Unpause'); |
|  | else $('#pause').html('Pause'); |
|  | } |

[**view raw**](https://gist.github.com/GleasonK/c4968791dab659a05804/raw/1f40cf26666a588e6379faa62f86df3e4da7a8de/09.js)[**09.js**](https://gist.github.com/GleasonK/c4968791dab659a05804#file-09-js) hosted with ❤ by [**GitHub**](https://github.com/)

These functions will change the buttons text depending on whether or not their stream is paused or volume is muted. However, both these functions also publish a control message to all the other users in your group chat. You can react to a user pausing their stream or muting their mic using the controllers ctrl.videoToggle and ctrl.audioToggled listeners.

Head back to your login function, where all phone and controller configurations were made. At the end of the function just before returning false, let’s add a video/audio toggle listener.

|  |  |
| --- | --- |
|  | function login(form){ |
|  | ... |
|  | ctrl.videoToggled(function(session, isEnabled){ |
|  | ctrl.getVideoElement(session.number).toggle(isEnabled); // Hide video is stream paused |
|  | }); |
|  | ctrl.audioToggled(function(session, isEnabled){ |
|  | ctrl.getVideoElement(session.number).css("opacity",isEnabled ? 1 : 0.75); // 0.75 opacity is audio muted |
|  | }); |
|  | return false; |
|  | } |

[**view raw**](https://gist.github.com/GleasonK/c4968791dab659a05804/raw/1f40cf26666a588e6379faa62f86df3e4da7a8de/10.js)[**10.js**](https://gist.github.com/GleasonK/c4968791dab659a05804#file-10-js) hosted with ❤ by [**GitHub**](https://github.com/)

You can see that both callbacks to the toggle listeners take two parameters, session, and isEnabled). That session is the same session from the receiver, which holds all information about the user who toggled a stream, and isEnabled is a boolean which tells if the stream is muted/paused or not.

For video, we simply get the video element and toggle it’s visibility with jQuery if a user pauses their stream. As for audio, we add a slight transparency to the video element, allowing you to immediately tell who is muted.

What Now?

This guide has only provided simple suggestions for your video chatting app. Get creative! These are very open callbacks. If you wanted to append a mic-off image when a user mutes, you can. Or change their video element to a photo when they pause streaming, easy.

Want to learn more?

Good, that never-ending quest for knowledge will get you far in life. Here are some other resources PubNub offers on WebRTC:

* [PubNub WebRTC SDK](https://github.com/stephenlb/webrtc-sdk)
* [What is WebRTC](https://www.pubnub.com/blog/2014-01-14-what-is-webrtc/)?
* [PubNub WebRTC Video Chat Demo](https://www.pubnub.com/developers/demos/webrtc/)

Read More: [JavaScript](https://www.pubnub.com/blog/tag/javascript/), [VOIP or WebRTC Signaling](https://www.pubnub.com/blog/tag/voip-or-webrtc-signaling/), [WebRTC](https://www.pubnub.com/blog/tag/webrtc/)