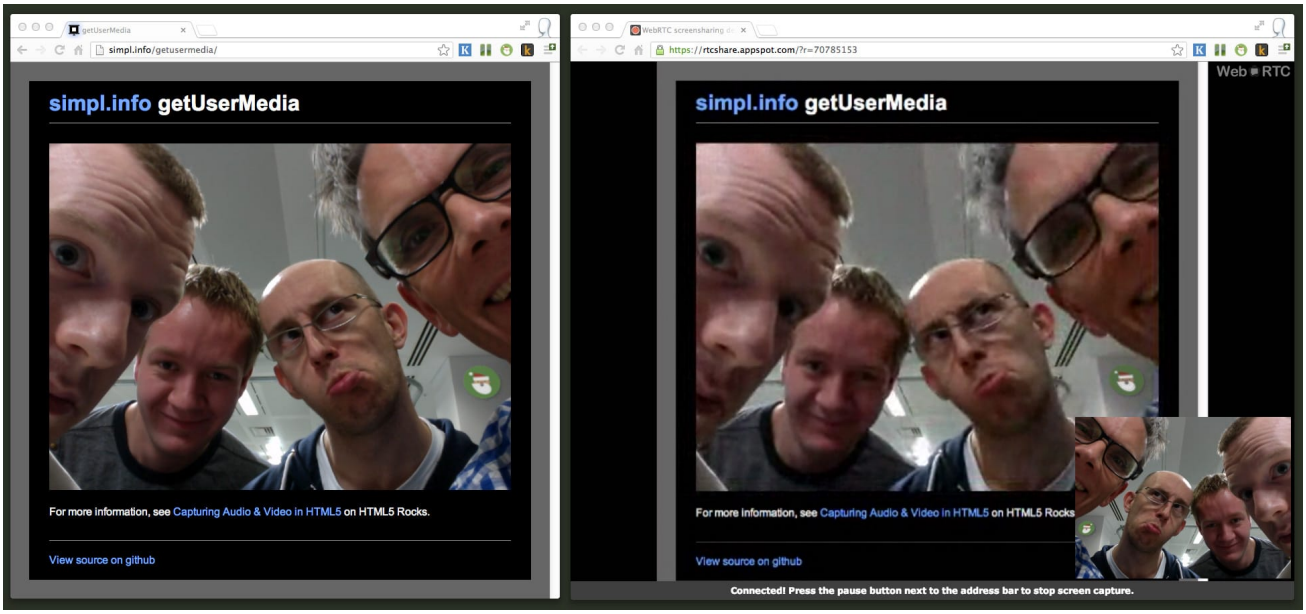
Screensharing with WebRTC

By Sam Dutton

Sam is a Developer Advocate

As we reported last week, there's been a **lot** happening lately with our old friend WebRTC.

Well... here's another first: WebRTC screensharing.



Here's a screencast: youtube.com/watch?v=tD0QtBUZsF4

...and here's the code: github.com/samdutton/rtcshare

In essence, we've built an experimental Chrome extension that uses `RTCPeerConnection` and `chrome.tabCapture` to share a live 'video' of a browser tab. If you want to try it out, you'll need Chrome Canary, and you'll need to enable Experimental Extension APIs on the *about:flags* page.

Our prototype relies heavily on the mighty apprtc.appspot.com demo and, to be frank, it's a bit of a hack! But... it's a proof of concept, and it works.

Here's how we did it:

1. When the user clicks the extension icon (the 'record button' next to the address bar), the extension's background script *background.js*, appends an iframe to itself, the `src` of which is rtcshare.appspot.com. In *background.js* it's only used to get values such as

`token` and `room_key`. We told you this was a hack :^}! This is a chopped and channeled version of apprtc.appspot.com. As with the apprtc example, rtcshare.appspot.com is also used for the remote client.

chrome.browserAction.onClicked.addListener(function(tab) { var currentMode = localStorage["capturing"]; var newMode = currentMode === "on" ? "off" : "on"; if (newMode === "on"){ // start capture appendIframe(); } else { // stop capture chrome.tabs.getSelected(null, function(tab){ localStream.stop(); onRemoteHangup(); }); // set icon, localStorage, etc. } }

2. When the iframe has loaded, *background.js* gets values from it (generated by the rtcshare.appspot.com app) and calls `chrome.tabCapture.capture()` to start capturing a live stream of the current tab.

   function appendIframe(){ iframe = document.createElement("iframe"); iframe.src="https://rtcshare.appspot.com"; document.body.appendChild(iframe); iframe.onload = function(){ iframe.contentWindow.postMessage("sendConfig", "*"); }; } // serialised config object messaged by iframe when it loads window.addEventListener("message", function(event) { if (event.origin !== "https://rtcshare.appspot.com"){ return; } var config = JSON.parse(event.data); room_link = config.room_link; // the remote peer URL token = config.token; // for messaging via Channel API // more parameter set from config); function startCapture(){ chrome.tabs.getSelected(null, function(tab) { var selectedTabId = tab.id; chrome.tabCapture.capture({audio:true, video:true}, handleCapture); // bingo! }); }

3. Once the live stream is available (in other words, a live 'video' of the current tab), *background.js* kicks off the peer connection process, and signalling is done via rtcshare.appspot.com using XHR and Google's Channel API. All in all, it works like the apprtc demo, except that the video stream communicated to the remote peer is from `chrome.tabCapture` and not `getUserMedia()`.

   function handleCapture(stream){ localStream = stream; // used by RTCPeerConnection addStream(); initialize(); // start signalling and peer connection process }

4. For demo purposes, this prototype extension opens a new tab with the URL provided by rtcshare.appspot.com, which has a 'room number' query string added. Of course, this URL could be opened on another computer, in another place, and THAT might be the start of something useful!

   chrome.tabs.create({url: room_link});

We envisage a lot of interesting use cases for screensharing and, even at this early stage of development, we're impressed at how responsive and stable plugin-free tab capture and sharing can be.

As ever, we welcome your comments: about this extension and about the WebRTC APIs in general. If you want to learn more about WebRTC, check out the HTML5 Rocks article or our Quick Start Guide.

Happy hacking -- and best wishes for 2013 from everyone at HTML5R and WebRTC!

Last updated July 2, 2018.