Recording Video from the User

By Mat Scales

Mat is a contributor to Web**Fundamentals**

Many browsers now have the ability to access video and audio input from the user. However, depending on the browser it might be a full dynamic and inline experience, or it could be delegated to another app on the user's device.

Start simple and progressively

The easiest thing to do is simply ask the user for a pre-recorded file. Do this by creating a simple file input element and adding an `accept` filter that indicates we can only accept video files and a `capture` attribute that indicates we want to get it direct from the camera.

```
<input type="file" accept="video/*" capture>
```

This method works on all platforms. On desktop it will prompt the user to upload a file from the file system (ignoring the `capture` attribute). In Safari on iOS it will open up the camera app, allowing you to record video and then send it back to the web page; on Android it will give the user the choice of which app to use record the video in before sending it back to the web page.

Many mobile devices have more than one camera. If you have a preference, you can set the `capture` attribute to `user`, if you want the camera that faces the user, or `environment` if you want the camera that faces outward.

```
<input type="file" accept="video/*" capture="user">
<input type="file" accept="video/*" capture="environment">
```

Note that this is just a hint - if the browser doesn't support the option, or the camera type you ask for isn't available, the browser may choose another camera.

Once the user has finished recording and they are back in the website, you need to somehow get ahold of the file data. You can get quick access by attaching an `onchange` event to the input element and then reading the `files` property of the event object.

```
<input type="file" accept="video/*" capture="camera" id="recorder">
<video id="player" controls></video>
<script>
  var recorder = document.getElementById('recorder');
  var player = document.getElementById('player')'

  recorder.addEventListener('change', function(e) {
    var file = e.target.files[0];
    // Do something with the video file.
    player.src = URL.createObjectURL(file);
  });
</script>
```

Once you have access to the file you can do anything you want with it. For example, you can:

- Attach it directly to a `<video>` element so that you can play it
- Download it to the user's device
- Upload it to a server by attaching to an `XMLHttpRequest`
- Draw the frames into a canvas and apply filters to it

Whilst using the input element method of getting access to video data is ubiquitous, it is the least appealing option. We really want to get access to the camera and provide a nice experience directly in the page.

Access the camera interactively

Modern browsers can have a direct line to the camera allowing us to build experiences that are fully integrated with the web page and the user will never leave the browser.

Acquire access to the camera

We can directly access the camera by using an API in the WebRTC specification called `getUserMedia()`. `getUserMedia()` will prompt the user for access to their connected microphones and cameras.

If successful the API will return a `Stream` that will contain the data from either the camera or the microphone, and we can then either attach it to a `<video>` element, attach it to a WebRTC stream, or save it using the `MediaRecorder` API.

To get data from the camera we just set `video: true` in the constraints object that is passed to the `getUserMedia()` API

```
<video id="player" controls></video>
<script>
  var player = document.getElementById('player');

  var handleSuccess = function(stream) {
    player.srcObject = stream;
  };

  navigator.mediaDevices.getUserMedia({ audio: true, video: true })
      .then(handleSuccess)
</script>
```

If you want to choose a particular camera you can first enumerate the available cameras.

```
navigator.mediaDevices.enumerateDevices().then((devices) => {
  devices = devices.filter((d) => d.kind === 'videoinput');
});
```

You can then pass the deviceId that you wish to use when you call `getUserMedia`.

```
navigator.mediaDevices.getUserMedia({
  audio: true,
  video: {
    deviceId: devices[0].deviceId
  }
});
```

By itself, this isn't that useful. All we can do is take the video data and play it back.

Access the raw data from the camera

To access the raw video data from the camera you can draw each frame into a `<canvas>` and manipulate the pixels directly.

For a 2D canvas you can use the `drawImage` method of the context to draw the current frame of a `<video>` element into the canvas.

```
context.drawImage(myVideoElement, 0, 0);
```

With a WebGL canvas you can use a `<video>` element as the source for a texture.

```
gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, myVideo
```

Note that in either case this will use the *current* frame of a playing video. To process multiple frames you need to redraw the video to the canvas each time.

You can learn more about this in our article about applying real-time effects to images and video.

Save the data from the camera

The easiest way to save the data from the camera is to use the `MediaRecorder` API.

The `MediaRecorder` API will take the stream created by `getUserMedia` and then progressively save the data from the stream to you preferred destination.

```
<a id="download">Download
<button id="stop">Stop
<script>
  let shouldStop = false;
  let stopped = false;
  const downloadLink = document.getElementById('download');
  const stopButton = document.getElementById('stop');

  stopButton.addEventListener('click', function() {
    shouldStop = true;
  })

  var handleSuccess = function(stream) {
    const options = {mimeType: 'video/webm'};
    const recordedChunks = [];
    const mediaRecorder = new MediaRecorder(stream, options);

    mediaRecorder.addEventListener('dataavailable', function(e) {
      if (e.data.size > 0) {
        recordedChunks.push(e.data);
      }

      if(shouldStop === true && stopped === false) {
        mediaRecorder.stop();
        stopped = true;
      }
    });

    mediaRecorder.addEventListener('stop', function() {
      downloadLink.href = URL.createObjectURL(new Blob(recordedChunks));
      downloadLink.download = 'acetest.webm';
    });

    mediaRecorder.start();
  };

  navigator.mediaDevices.getUserMedia({ audio: true, video: true })
      .then(handleSuccess);

</script>
```

In our case we are saving the data directly into an array that we can later turn in to a `Blob` which can be then used to save to our Web Server or directly in storage on the user's device.

Ask permission to use camera responsibly

If the user has not previously granted your site access to the camera then the instant that you call `getUserMedia` the browser will prompt the user to grant your site permission to the camera.

User's hate getting prompted for access to powerful devices on their machine and they will frequently block the request, or they will ignore it if they don't understand the context of which the prompt has been created. It is best practice to only ask to access the camera when first needed. Once the user has granted access they won't be asked again, however, if they reject access, you can't get access again to ask the user for permission.

Warning: Asking for access to the camera on page load will result in most of your users rejecting access.

Use the permissions API to check if you already have access

The `getUserMedia` API provides you with no knowledge of if you already have access to the camera. This presents you with a problem, to provide a nice UI to get the user to grant you access to the camera, you have to ask for access to camera.

This can be solved in some browsers by using the Permission API. The `navigator.permission` API allows you to query the state of the ability to access specific API's without having to prompt again.

To query if you have access to the user's camera you can pass in `{name: 'camera'}` into the query method and it will return either:

- `granted` — the user has previously given you access to the camera;

- `prompt` — the user has not given you access and will be prompted when you call `getUserMedia`;
- `denied` — the system or the user has explicitly blocked access to the camera and you won't be able to get access to it.

And you can now check quickly check to see if you need to alter your user interface to accommodate the actions that the user needs to take.

```
navigator.permissions.query({name:'camera'}).then(function(result) {
  if (result.state == 'granted') {

  } else if (result.state == 'prompt') {

  } else if (result.state == 'denied') {

  }
  result.onchange = function() {

  };
});
```

Last updated July 2, 2018.