

Handed to Prof. Ari Rappoport

By Akiva Hassab

ID 209023605

Extension's summary:

Equalization is the process of adjusting the balance between frequency components within an electronic signal. The most well-known use of equalization is in sound recording and reproduction but there are many other applications in electronics and telecommunications.

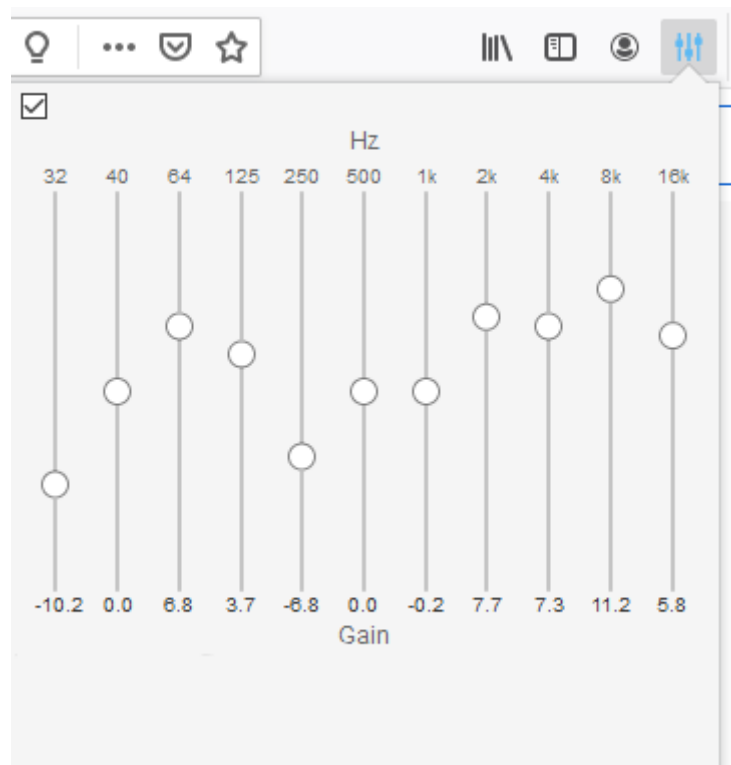
Audio Equalizer allows improving the sound quality to get more pleasure from listening to music and watching videos. The most basic type of equalization is the bass control on the old home audio equipment to control low frequencies and the treble control to adjust high frequencies. This extension adds a new button to the Firefox toolbar. When the user clicks the button, we display a popup enabling them to choose a wide set of frequency. These frequencies be broadly described as “Bass” (32, 64, 125), “Mid Range”(250, 500, 1K, 2K), & “Treble”(4K, 8K, 16K).

Good to know: These numbers represent octaves & each slider is 1 octave apart.

the extension lets the user easily adjust audio settings, the balance between frequency components in an audio file.

The interaction with the user:

The extension adds a toolbar button to the browser which controls the output of all media elements that are responsible for generating. Once you click on the equalizer button, a drop-down list will be shown with 10 sliders each one represents frequency. Using it the user can enable and disable the audio equalization.




Installing – to install the extension, one simply need to load the manifest.json file from the about:debugging section (pressing on Load Temporary Add-on button).

implementation

I define a browser action, which is a button attached to the Firefox toolbar.

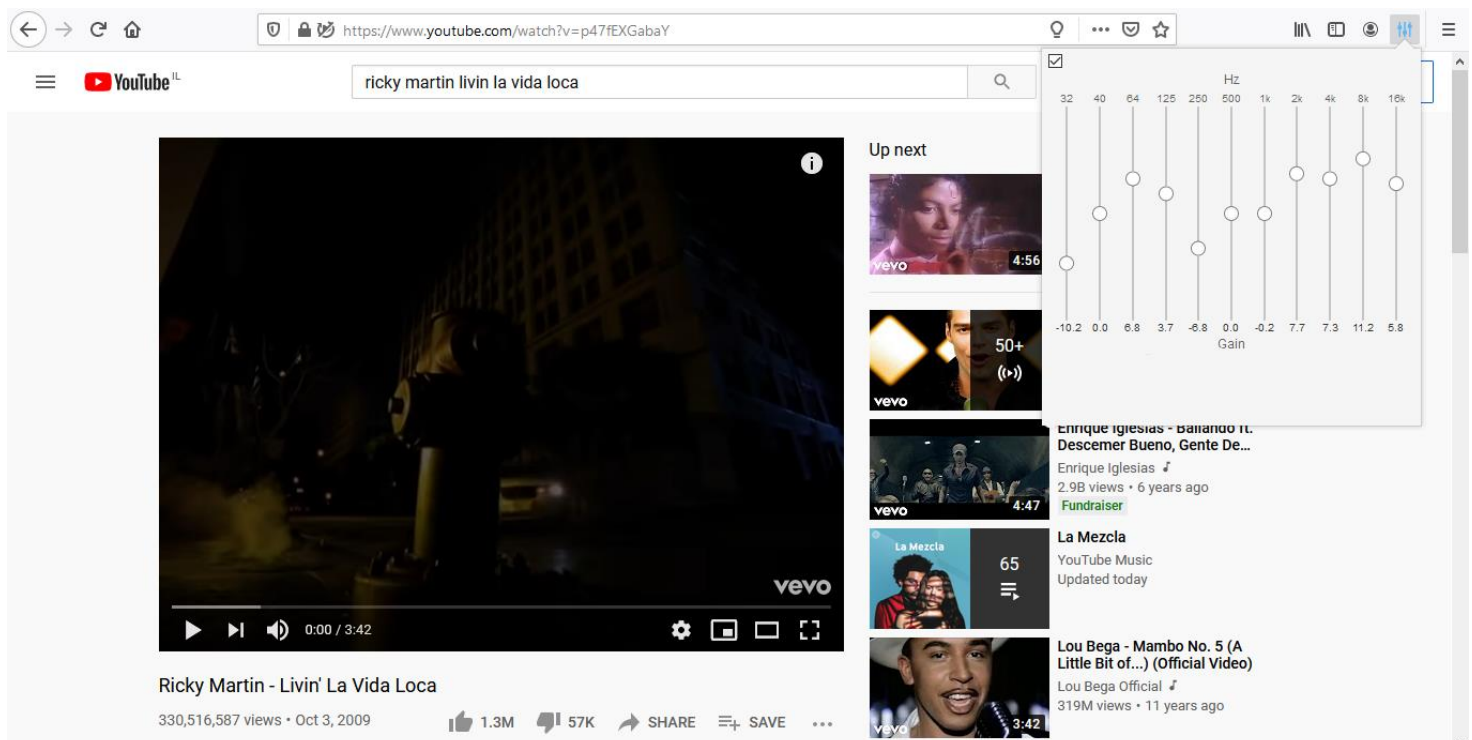
For the button we'll supply:

- an icon 
- a popup to open when the button is pressed. The popup will include HTML, CSS, and JavaScript.

The popup

the popup will consist of three files:

- **popup.html** defines the content of the panel
- **popup.css** styles the content
- **popup.js** handles the user's choice by running a content script in the active tab



The Audio Equalizer algorithm:

abstract: Located in the backend.js file and equalizer.js. When getting an order to reducing (attenuating) the levels of different frequencies of media elements in web pages. the algorithm first gets the new gain parameter chosen by the user in the UI then gets all the video/audio fields in the current webpage. Then for each of the elements, the algorithm updates the gain of the chosen frequency.

More details

The Equalizer in the extension represent by a map which the keys is the frequency and the value is gains

```
const state = {  
  
  gains: {  
    32: 0.0,  
    64: 0.0,  
    125: 0.0,  
    250: 0.0,  
    500: 0.0,  
    1000: 0.0,  
    2000: 0.0,  
    4000: 0.0,  
    8000: 0.0,  
    16000: 0.0  
  },  
}
```

When getting an order to reducing the levels of different frequencies We get from the [<body>](#) or [<frameset>](#) node of the current document all HTML Video/audio element (<video>/ <audio>) using querySelectorAll witch return [NodeList](#) representing a list of the document's elements that match the specified group of selectors.

Then for each HTML Video/audio element from above I create a new `MediaElementAudioSourceNode` object . Using `createMediaElementSource()` method which used to given an existing HTML `<audio>` or `<video>` element, the audio from which can then be played and manipulated.

The `MediaElementAudioSourceNode` interface represents an audio source consisting of an HTML5 `<audio>` or `<video>` element. It is an `AudioNode` that acts as an audio source. (in our presentation Omer and I explained it)

Then to update\ filter the signal (our audio), we use the `createBiquadFilter()`.

The `createBiquadFilter()` method of the [BaseAudioContext](#) interface creates a [BiquadFilterNode](#), which represents a second-order filter configurable as several different common filter types. The `BiquadFilterNode` interface represents a simple low-order filter. It is an [AudioNode](#) that can represent different kinds of filters, tone control devices, and graphic equalizers.

The [BiquadFilterNode](#) has many Properties the most important for us is the frequencies which are constant (we choose them already) and the gain value which we update to the new gain that the user chose.

Connection

To make a connection between the different contexts inside the extension I use `runtime.connect()` which return Port through which messages can be sent and received.

Message keys:

```
UPDATE_GAIN: 'update::gain',  
SET_GAIN: 'set::gain',  
QUERY_STATE: 'query::state',
```

- UPDATE_STATE - the user choose a new state need to update by using Audio Equalizer algorithm
- SET_GAIN – the gain was update
- QUERY_STATE- default state

When the user change the one band in the GUI we handle the New State then send a UPDATE_GAIN message which it's a index to update the audio signal with the new state using `.runtime.sendMessage()` .

It's a simple extension, but shows many of the basic concepts of the WebExtensions API:

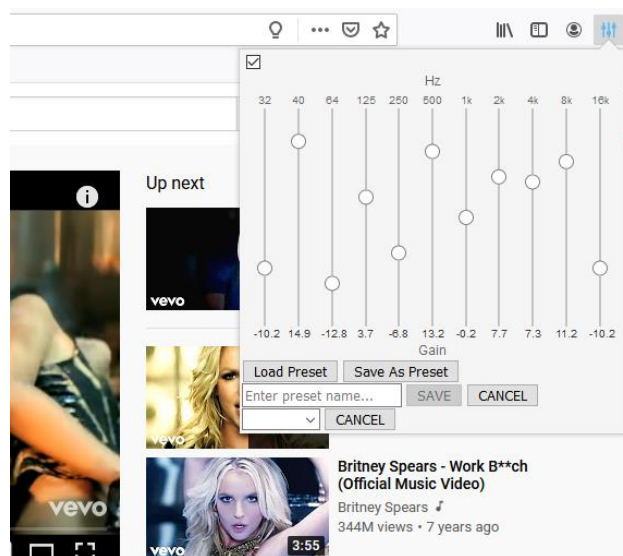
- write a browser action with a popup
- defining a popup panel using HTML, CSS, and JavaScript
- use web accessible resources to enable web pages to load packaged content
- give the popup style and behavior using CSS and JS
- injecting content scripts into web pages
- Works with HTML5 audio and video element.
- communicating between content scripts and the rest of the extension
- how to have different browser_action images based upon the theme
- send a message from the main extension to a content script
- adding a button to the toolbar

Extra point:

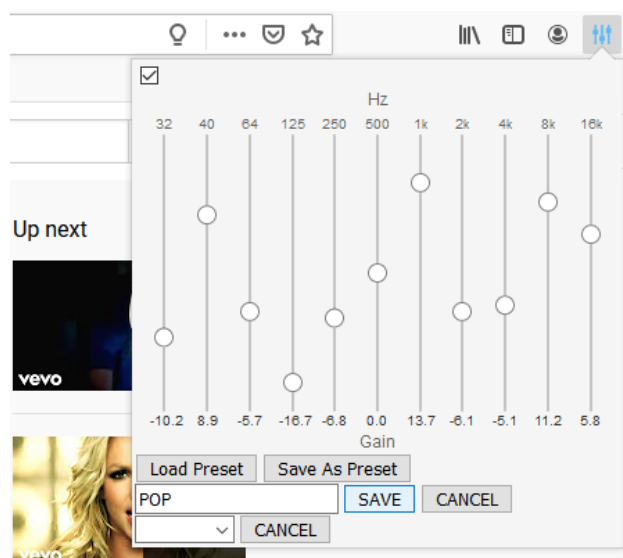
Another extension I did, for the extra point. It's an option to save certain preset that the user made, this give a chance to the user to use his patterns in the future or to try the patterns on a lot more videos .

Steps:

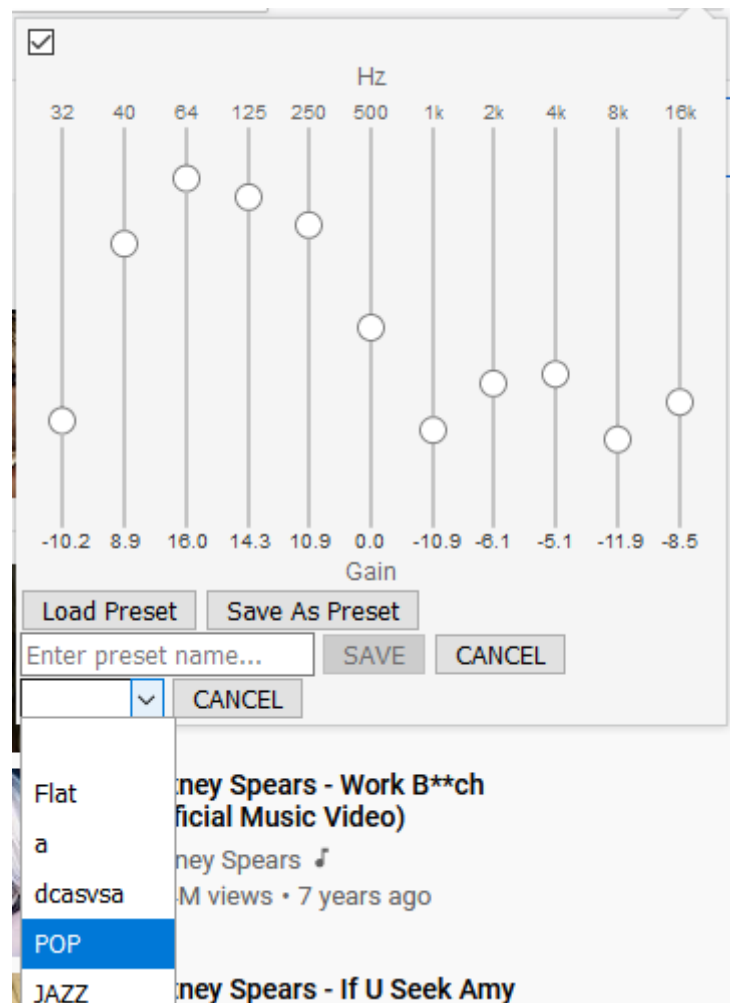
1.Choose the preset



2.choose a name to the preset and save it



3. use a store preset



To implement this extension list of maps and each preset that I Want to save I saved its map.

Storage: All the data is stored locally using the Firefox extension storage, to access the data, I use `browser.storage.local`