browser.extend('awe')

Building Browser Extension

Viswaprasath

Building Browser Extension

Get Started with Cross Browser WebExtension Development.

Viswaprasath

This book is for sale at http://leanpub.com/mozwebext

This version was published on 2018-11-29

Tweet This Book!

Please help Viswaprasath by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I just got @MozWebExt book authored by @iamvp7 along with @mozamo contributors.. Feeling excited to learn about Browser Customization. Excited for #WebExtLearn Journey

The suggested hashtag for this book is #WebExtLearn.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#WebExtLearn

Contents

Acknowledgments i

Who should read this book ii

About Browser Extensions iii

How is this book organised iv

Getting Started with Mozilla Webextensions 1
- Why Web Extensions 3
- References 3

Running WebExtension in Firefox 4
- Step by Step procedure for running WebExtension 4
- References 9

Experiment 1: Building Motivational Tab 10
- Problem Statement and Solution 10
- Building Blocks of Our Motivation Tab WebExtension 11
- Exercise 14
- References 14

Experiment 2: Building Power Search Add-on 15
- Problem Statement and Solution 15
- Building Blocks of Our Power Search Add-on 16
- Assembling our Power Search Extension Parts 19
- Exercise 23
- References 23

Experiment 3: Building Tabs Closer 24
- Problem Statement and Solution 24
- Building Blocks of Our Social Media Tab Closer 25
- Building Blocks of All Tabs Closer 27
- Exercise 31
- References 32

Experiment 4: Building PDF WebPage Extension **33**
Problem Statement and Solution 33
Building Blocks of PDF WebPage Extension 33
Exercise 40
References 40

Experiment 5: Building Water Notifier Extension **41**
Problem Statement and Solution 41
Building Blocks of Our Water Notifier Extension 41
Exercise 46
References 46

Experiment 6: Building Shortcuts for Youtube Controller **47**
Problem Statement and Solution 47
Building Blocks of Our Youtube Controller Extension 47
Assembling our Power Search Extension Parts 49
Exercise 51
References 52

Experiment 7: Building Quick Stackoverflow searcher **53**
Problem Statement and Solution 53
Building Blocks of Our Quick Stackoverflow searcher 53
Assembling our Power Search Extension Parts 54
Exercise 58
References 58

Publishing our First WebExtension **60**
Congrats 65
Exercise 65

Acknowledgments

I would like to thank my parents first without whom I might have not been able to achieve anything in this world.

Second I would like to thank my friends and family members who supported me throughout this journey.

Special thanks to Amy and Caitlin from AMO team at Mozilla who supports contributors, encourages and motivate us to learn many things and contribute back to the community. I am very proud to say without their help it might have been difficult to stick with the community.

I also would like to thank Trishul, Santhosh, and Karthic the rockstar contributors who helped all these days and was together as a team to build community in India. Without these guys, learning won’t be that much fun.

Who should read this book

This book is targeted mainly for fresh developers who have a basic understanding of HTML, CSS, and Javascript or developers who would like to get started with developing browser extensions. With this book, you will learn how to build simple extensions using tab manipulation. My strict advice is to learn a bit of Javascript so using this book will be easier. In the future, I am targeting another book which will be targeted towards a wider audience and we will be learning more.

About Browser Extensions

This book totally consists of seven experiments. Each experiment is individual extension which can be used separately in your browser.
A browser extension (or “add-on”) is code that can add features or functionality to Firefox. It can be plugged at any time and can be removed if not required. Extensions for Firefox are written using the WebExtensions API, which to a large extent are supported across different browsers like Mozilla Firefox, Google Chrome, Opera Browser, and Microsoft Edge.

How is this book organised

The first two chapters **Getting Started with the WebExtensions API** and **Running Extensions in Firefox** will help you learn about the basics of browser extensions and teach you how to configure your browser for developing the extension and how to run them locally.

In *Experiment 1*, you will learn how to create very basic simple extension without much need of Javascript. If you are learning JS for the first time, you can work on this experiment.

In *Experiment 2*, we will be getting started with the first user interface (UI) with related to browser extensions. We will be getthe search engines in our browser and create a context menu and do the interaction with them. Context menus are most used UI in the browser by average users.

In *Experiment 3*, we will learn about the second type of user interface for extensions. We will learn about browserAction, pop-up HTML pages, and interactions between the page and background script.

In *Experiment 4*, we will learn about the third type of user interface for xtensions. We will learn about page actions and their interactions between the page and background script. This is similar to browserAction but with minimal difference.

In *Experiment 5*, we will be learning about notifications in extensions. This is one of the most interesting experiment and is my favorite one. This is very easy and most enjoyable experiment among all.

In *Experiment 6*, we will be learning how to build our own shortcuts using WebExtension API which works inside the browser. All the experiment before this we used to have some sort of visible user interface which can be displayed, but in this experiment, we will be doing everything in the background.

In *Experiment 7*, this is the final experiment in this book. We will be learning how to interact with the address bar of the browser extension.

The real journey starts at the end of these 7 experiments. These experiments will be more like the stepping stones of our extenension development.

Getting Started with Mozilla Webextensions

During 2015 Firefox team has made an exciting very big announcement to Firefox add-ons, they have brought a new extension API's called `WebExtension` [1]. This update has a lot of exciting announcements for developers.

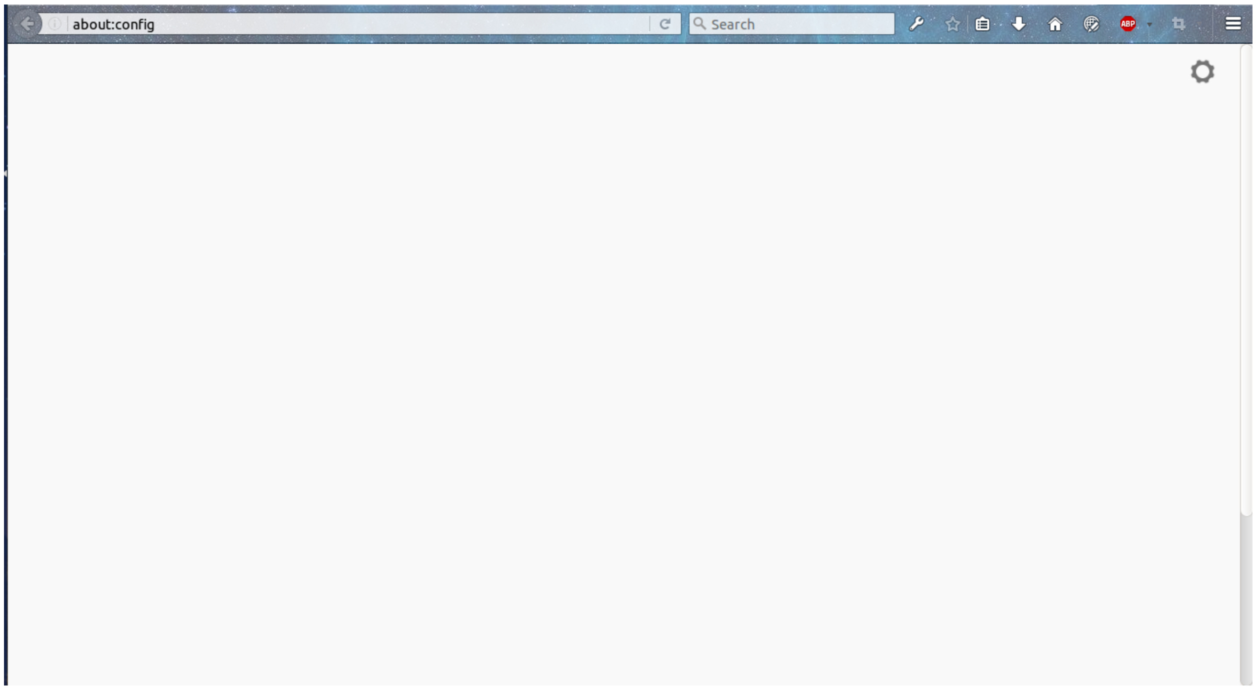
1. The review process will become faster
2. Development is easier, and the Addons will be compatible with Chrome and Opera.

You can find the whole information regarding this update in `Add-ons Blog` [2]

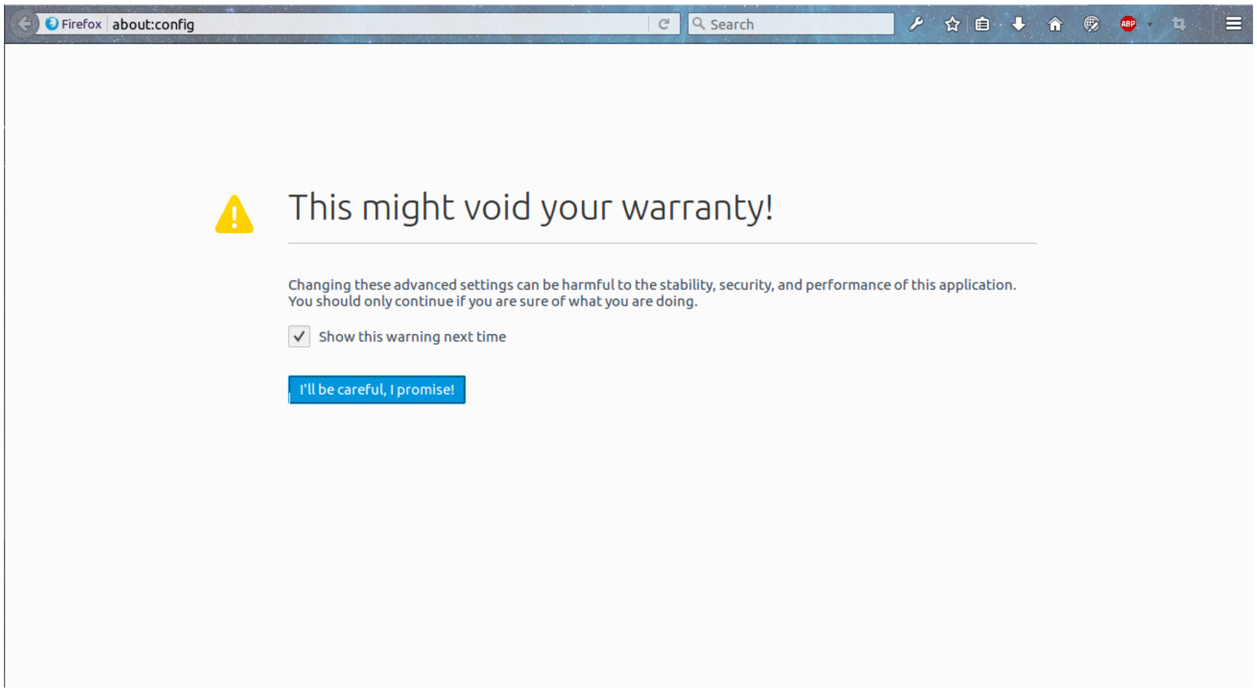
The next step is to download the Firefox Browser and install them. You can either go for `developer Edition` [3] or `Nightly` [4]

So as of now, Firefox Will allow only the add-ons which are signed when you start the browser. So for development purpose, we have to change the settings.

1. Open the browser
2. in the URL bar type about:config



config in URL bar



This might void your warranty!

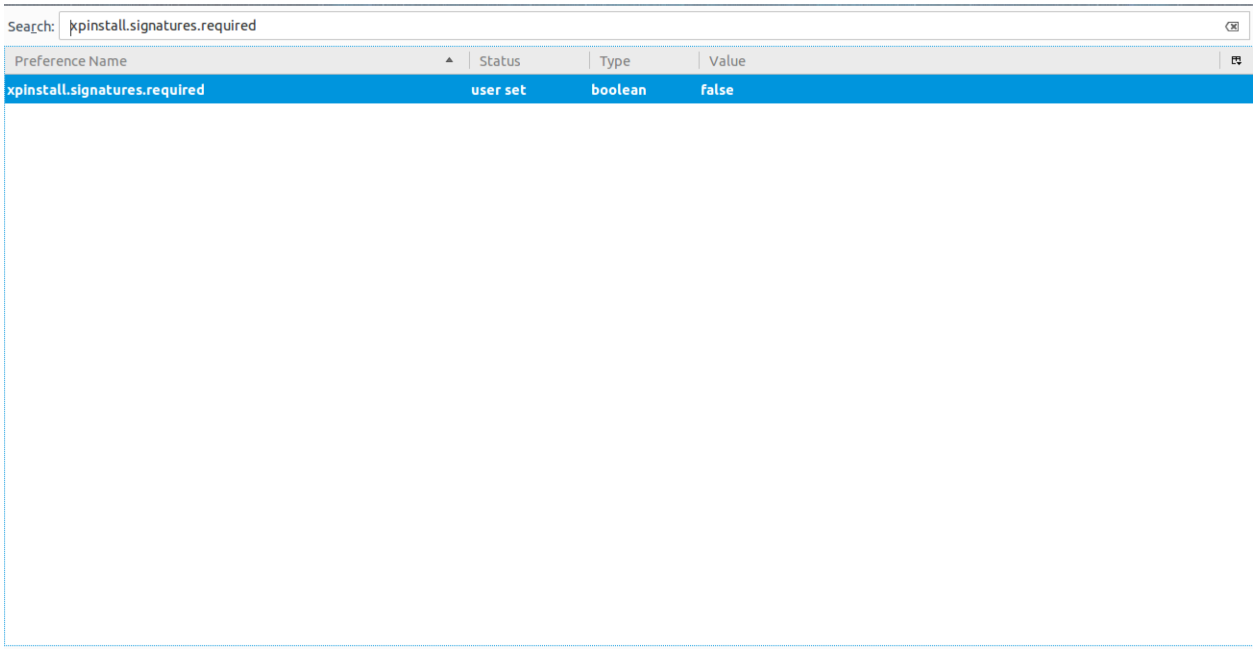
Changing these advanced settings can be harmful to the stability, security, and performance of this application. You should only continue if you are sure of what you are doing.

[✓] Show this warning next time

I'll be careful, I promise!

Warning

1. In the search bar of that page type xpinstall.signatures.required, you will see the image as like below. By default, it will be true



Search: xpinstall.signatures.required

Preference Name	Status	Type	Value
xpinstall.signatures.required	user set	boolean	false

Signature

1. You can double click on it or else right click and select toggle. So it will be set to false

Why Web Extensions

One of the Main reason, to get started with Web Extensions is easier to develop. Previously I have tried developing some Chrome Extensions during my free time, so since the same thing is used here, it will be very easier for me to get started and develop well. So I write once for Firefox Add-ons and it will support multiple platforms.

Very excited for this new journey of learning.

References

- [1] https://wiki.mozilla.org/WebExtensions
- [2] https://blog.mozilla.org/addons/2015/08/21/the-future-of-developing-firefox-add-ons/
- [3] https://www.mozilla.org/en-US/firefox/developer/
- [4] https://nightly.mozilla.org/

Running WebExtension in Firefox

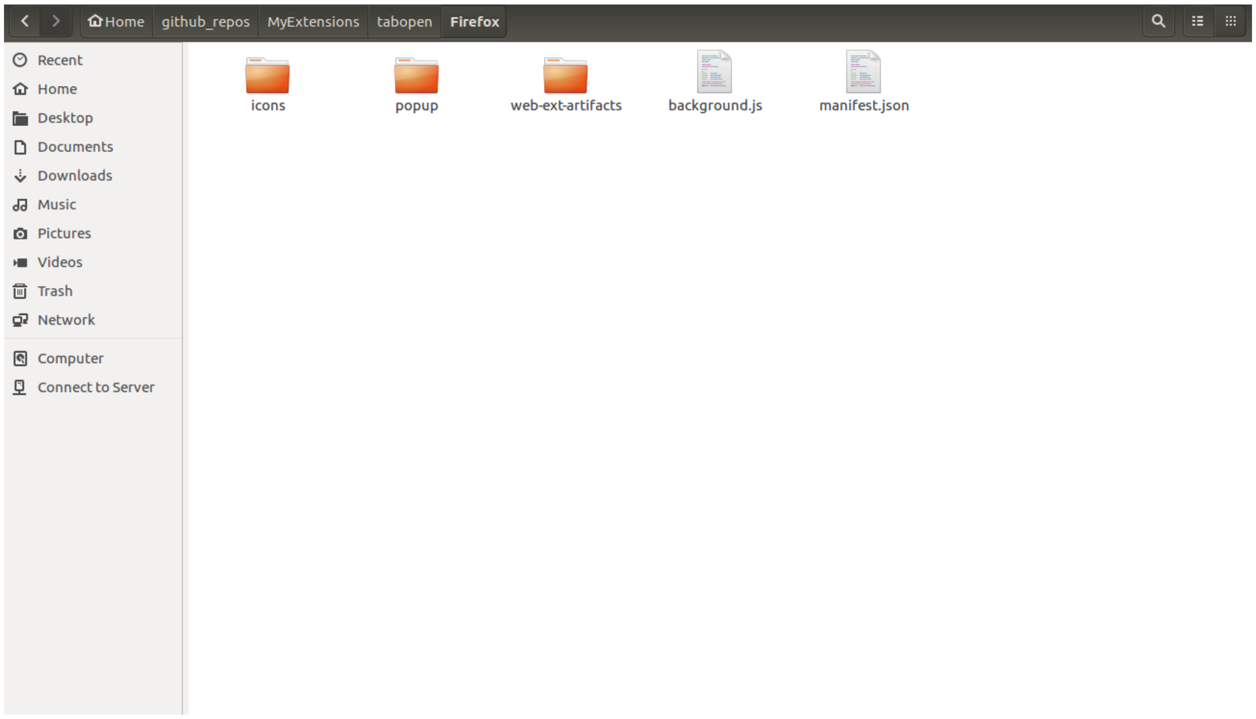
In this section, we will be exploring more how to run a Firefox WebExtensions.

I will be showing this demo with the help of the small extension I have written before, feel free to download it from `GitHub` [1], you can clone whole `repo` [2]. I personally use `web-ext` [3] tool for packing my extension, but for basic learning, it is not mandatory.

Step by Step procedure for running WebExtension

Step 1: Go to the directory which has contents of Extension

In my case, it will look like the below. The content showed below is the contents of Extension, which the link is shared above.

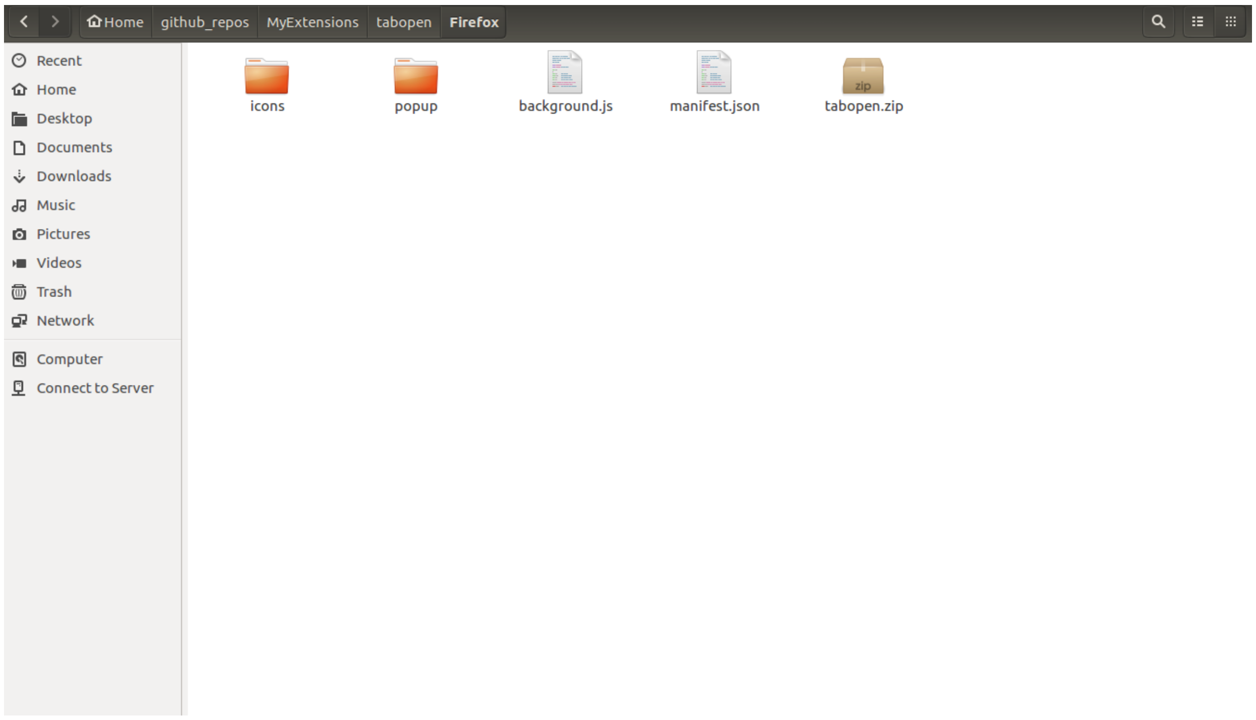


Directory Image

Step 2: We have to compress all the contents into Zip

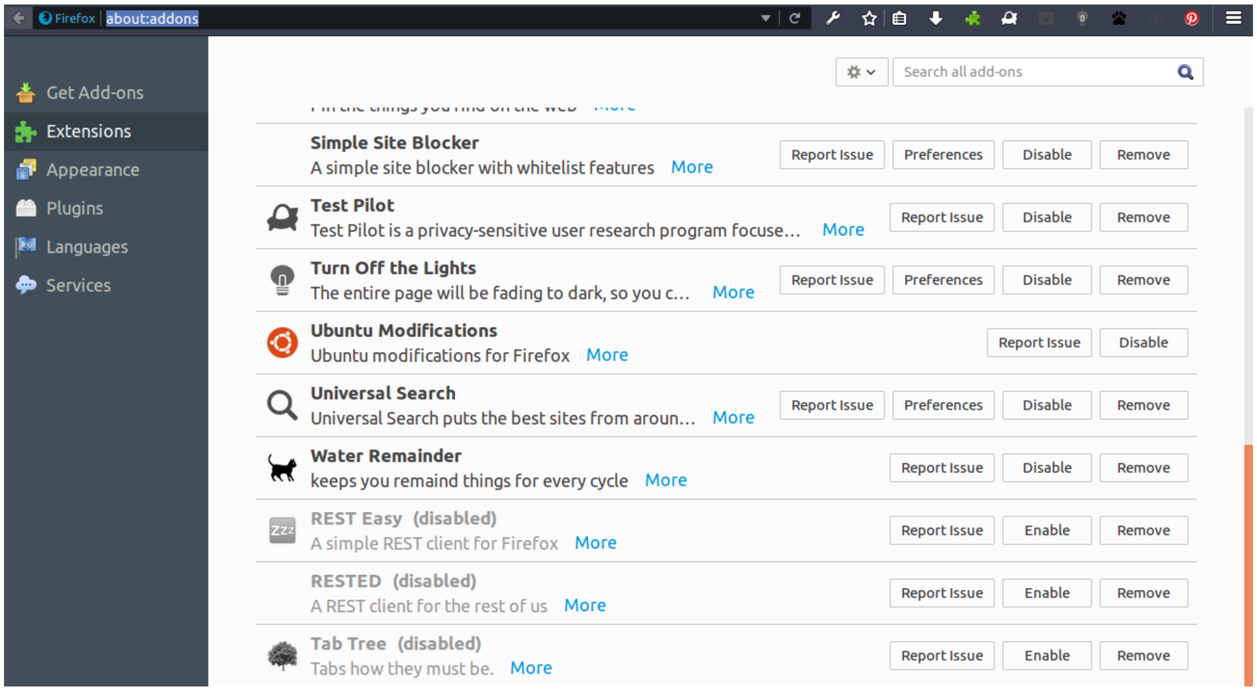
- Select all the folders and files

- Make it to a zip, anyname.zip



Directory Image

Step 3: Open about:addons or Ctrl+Shift+A



Simple Site Blocker
A simple site blocker with whitelist features

Test Pilot
Test Pilot is a privacy-sensitive user research program focuse…

Turn Off the Lights
The entire page will be fading to dark, so you c…

Ubuntu Modifications
Ubuntu modifications for Firefox

Universal Search
Universal Search puts the best sites from aroun…

Water Remainder
keeps you remaind things for every cycle

REST Easy (disabled)
A simple REST client for Firefox

RESTED (disabled)
A REST client for the rest of us

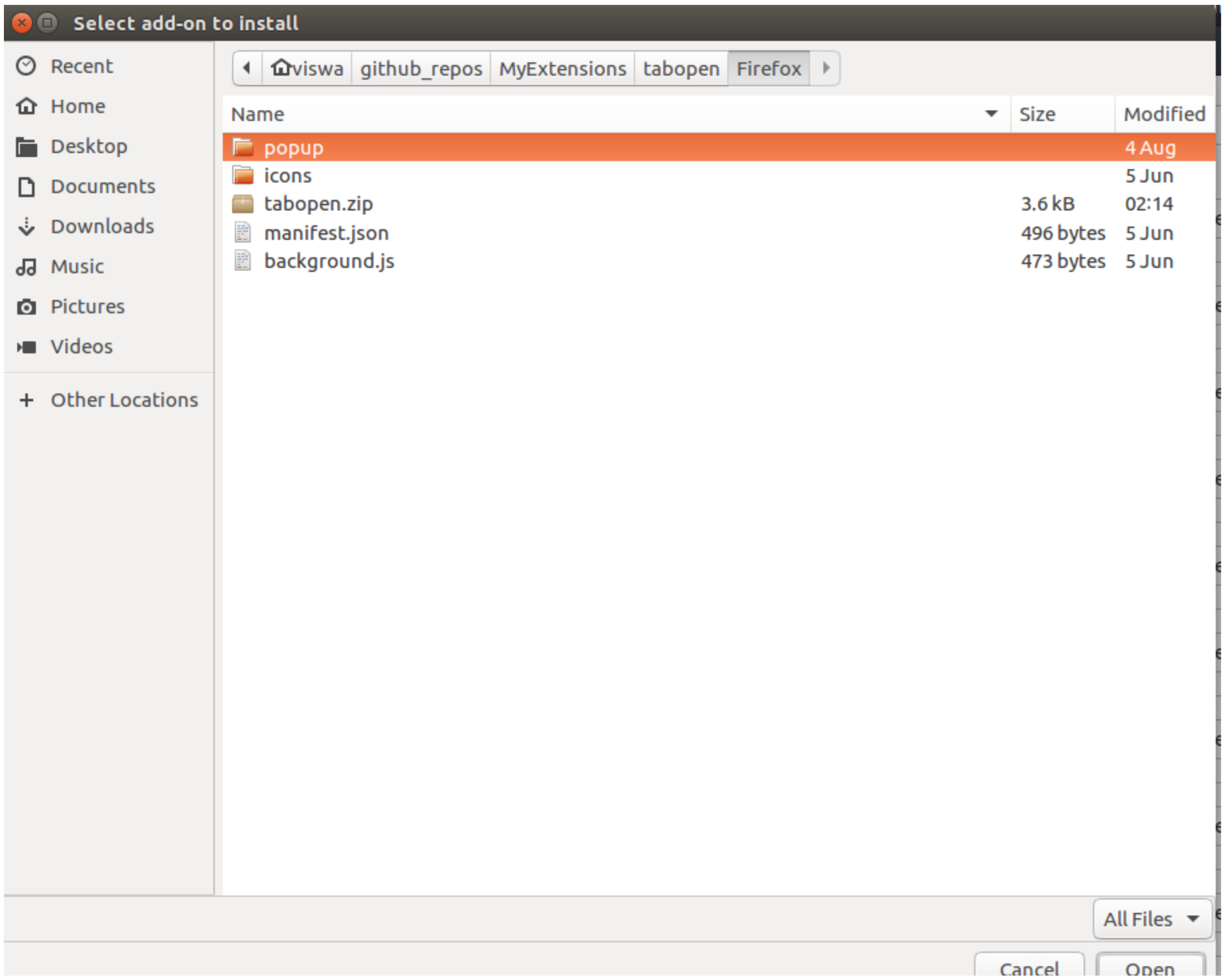
Tab Tree (disabled)
Tabs how they must be.

Directory Image

Step 4: Install Addons or Debug Addons

Install Addons

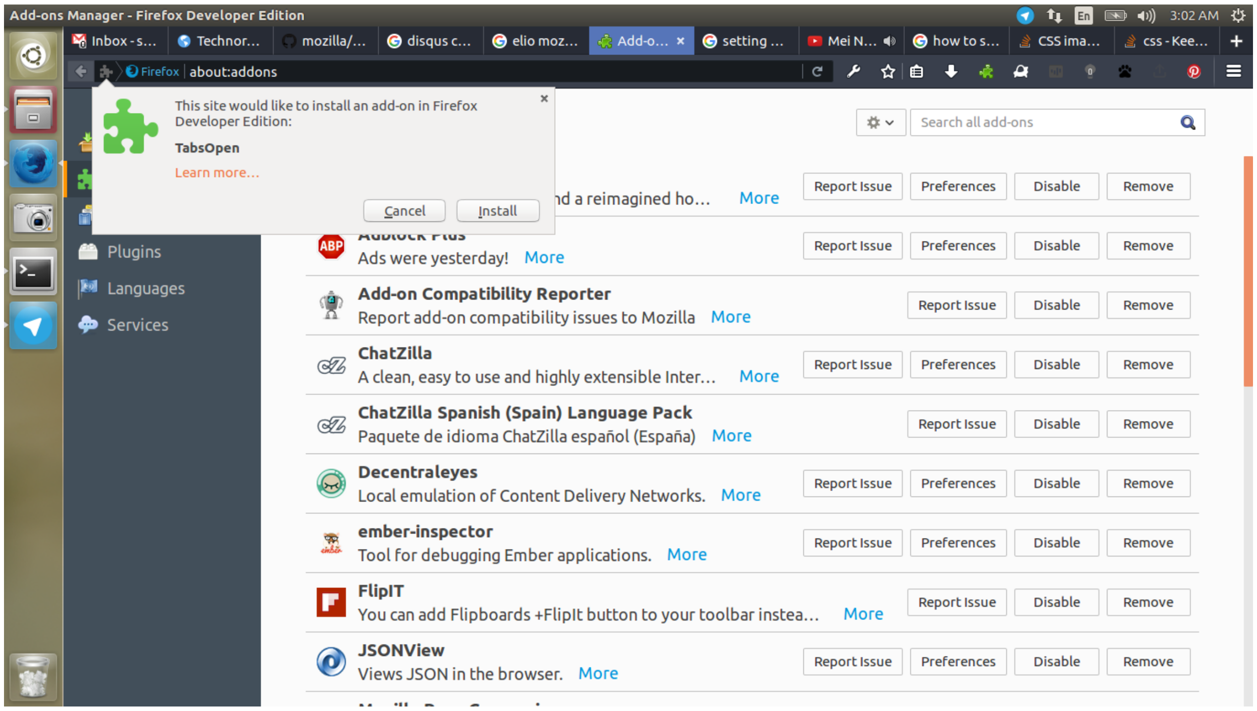
- Click on Tools for all add-ons (the icon - which is like Setting icon)
- Select Install Add-on From File ..
- Make sure you have set the file type to all files
- Choose the Zip which you created in step 2



Name	Size	Modified
popup		4 Aug
icons		5 Jun
tabopen.zip	3.6 kB	02:14
manifest.json	496 bytes	5 Jun
background.js	473 bytes	5 Jun

Choose the Zip file

- You will get a pop-up notification after selecting at left croner


Add-ons Manager - Firefox Developer Edition

This site would like to install an add-on in Firefox Developer Edition:
TabsOpen
Learn more...
Cancel Install

...nd a reimagined ho... More — Report Issue, Preferences, Disable, Remove

Adblock Plus
Ads were yesterday! More — Report Issue, Preferences, Disable, Remove

Add-on Compatibility Reporter
Report add-on compatibility issues to Mozilla More — Report Issue, Disable, Remove

ChatZilla
A clean, easy to use and highly extensible Inter... More — Report Issue, Preferences, Disable, Remove

ChatZilla Spanish (Spain) Language Pack
Paquete de idioma ChatZilla español (España) More — Report Issue, Disable, Remove

Decentraleyes
Local emulation of Content Delivery Networks. More — Report Issue, Preferences, Disable, Remove

ember-inspector
Tool for debugging Ember applications. More — Report Issue, Preferences, Disable, Remove

FlipIT
You can add Flipboards +FlipIt button to your toolbar instea... More — Report Issue, Disable, Remove

JSONView
Views JSON in the browser. More — Report Issue, Preferences, Disable, Remove

Choose the Zip file

- In that pop-up click Install button
- Now your Extension is installed.

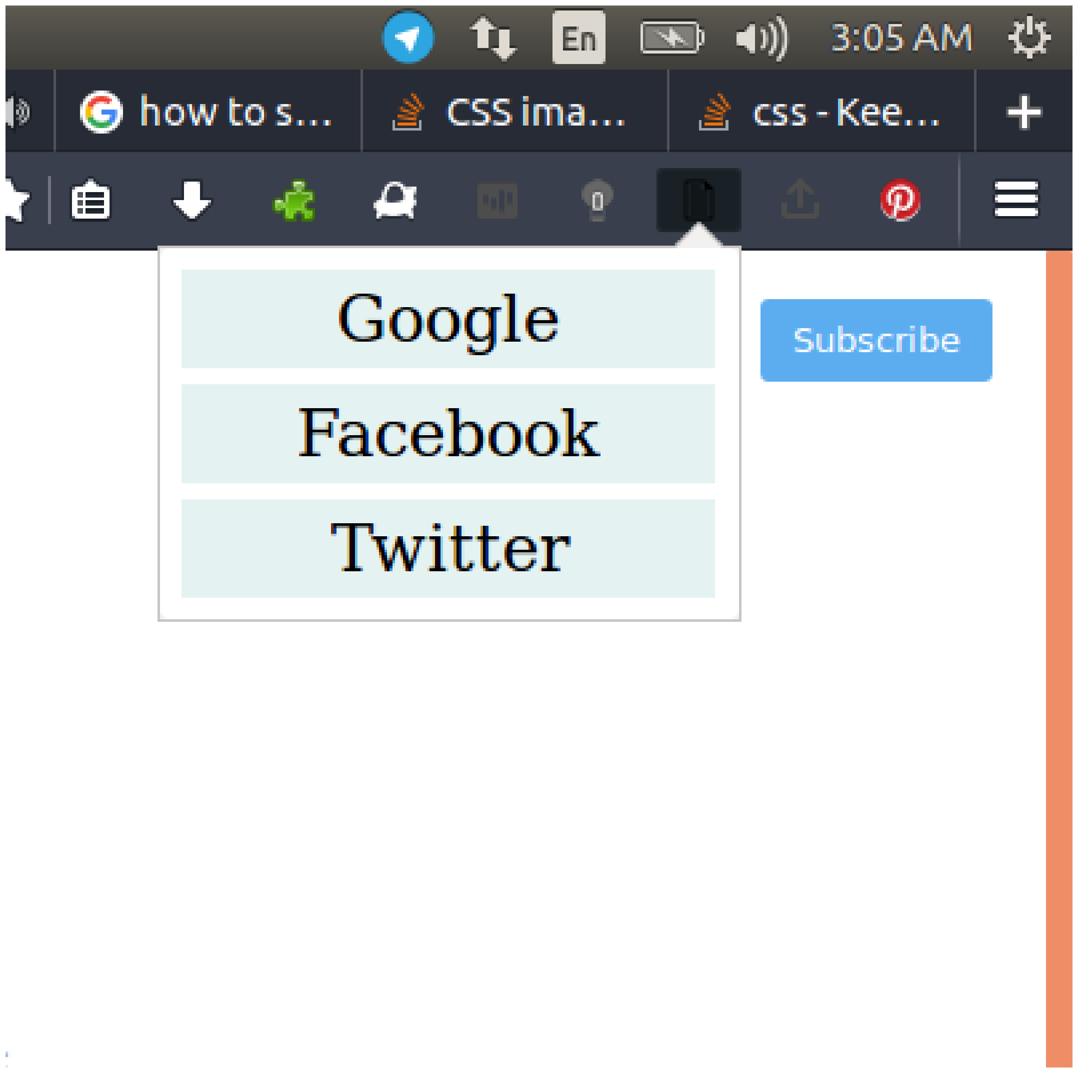
Debug Addons

During testing, it is enough to go with Debug Add-ons option. The only drawback will be, if we close the Firefox then the Add-on which we installed during our development phase will be uninstalled. Follow the steps below.

- Click on Tools for all add-ons (the icon - which is like Setting icon)
- Debug Add-on.
- Goto the folder where you source code is present.
- Select the manifest.json file, so the Add-on is installed temporarily.

Step 5: Run the Extension

For most of the Extensions you will get the icon in the toolbar itself. So for this, a page like an icon will be coming at the toolbar, if every code is run properly. Click on that icon.



Choose the Zip file

So on Selecting any one of the three websites, it will open a new tab and open the site. As of now, it is in static mode.

In future, based on user usage we can show the top 3 websites there.

References

- [1] https://github.com/iamVP7/MyExtensions/tree/master/tabopen/Firefox
- [2] https://github.com/iamVP7/MyExtensions/
- [3] https://github.com/mozilla/web-ext

Experiment 1: Building Motivational Tab

We got to know what is meant by WebExtension and how to install WebExtension from the source code available in our system. Those lessons are very important to us, as we move forward with our development, we will be always using them.

Our Approach is to learn the basic code, try them out with little modification and run them in our system, try out the exercise and share about this learning in the social medium of community benefit.

Problem Statement and Solution

Problem Statement

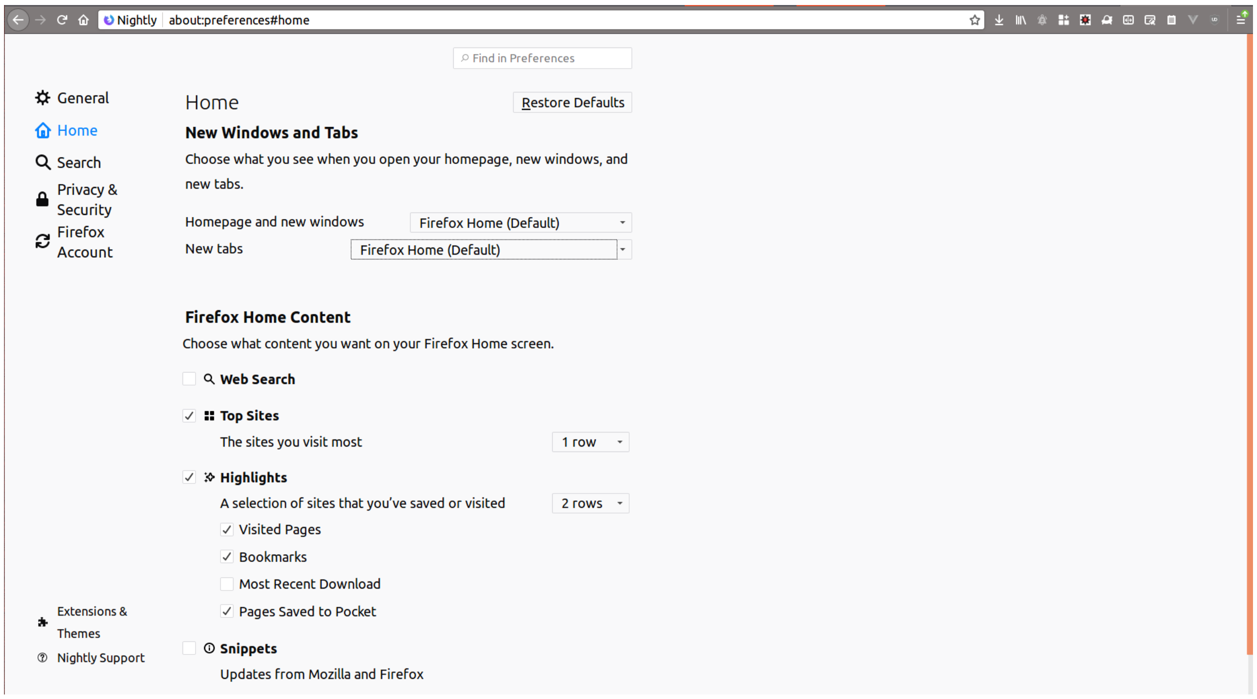
We have to open different web pages each and every day. Sometime we will be bored to do them. At that time, we may feel to get a cup of coffee. For sure it's not possible to count the number of copies we had. Having too much coffee or other caffeine products is injurious to health.

Solution

So the very first solution is to make sure we are not bored, and we are motivated when we open a new tab. We can be motivated by seeing strong quotes by our favorite personality or our family pictures whenever we open a new tab.

What is relationship with WebExtension???

Whenever you are opening a new tab, it means we are calling some internal API to open the tab. Each and every action we do is an API, some of them are exposed via WebExtension API. So by default, if you check in your browser you will understand you have options to change what should be shown when a new tab is opened. A screenshot from Firefox is shown below, check under the heading *New Windows and Tabs*



Home

New Windows and Tabs

Choose what you see when you open your homepage, new windows, and new tabs.

Homepage and new windows: Firefox Home (Default)

New tabs: Firefox Home (Default)

Firefox Home Content

Choose what content you want on your Firefox Home screen.

- [ ] **Web Search**
- [✓] **Top Sites**
  The sites you visit most: 1 row
- [✓] **Highlights**
  A selection of sites that you've saved or visited: 2 rows
  - [✓] Visited Pages
  - [✓] Bookmarks
  - [ ] Most Recent Download
  - [✓] Pages Saved to Pocket
- [ ] **Snippets**
  Updates from Mozilla and Firefox

New tab Configuration

How we can capture this

We need not worry much. It is easy to capture when a new tab is opened. And also it is easy to open our preferred page whenever a new tab or window is opened. In this section, we will be seeing how to open our own page with some random predefined quotes using *chrome_url_overrides* WebExtension API.

Building Blocks of Our Motivation Tab WebExtension

Writing manifest.json

For every program we will start by writing the manifest.json, as it will be giving us the clear view what we are going to do and what are all the components like permission, background script, resources and so on, we are going to have.

```
{
"manifest_version": 2,
"name" : "Motivation Tab",
"description" : "Shows Random Quotes ",
  "version": "1.0",
  "chrome_url_overrides" : {
    "newtab": "my-new-tab.html"
  }
}
```

The above is our manifest.json required for this program. The very important keys which we have to define for the manifest json are as follows

- name
- description
- manifest_version is always 2 (till Mozilla changes)
- version is the version of this addon, and it should be incremental.

Additionally for this particular Add-on (WebExtension) we are going to use *chrome_url_overrides*

What is chrome_url_overrides ??

We can remember it as URL Overrider, means it will be overriding the default with the value we give in the extension. In our example, we are saying to override the **newtab**, so whenever there is a new tab created if our extension is installed then, our extension will be overriding the other pages and will show the page which we have defined.

So our definition will be like below,

```
"chrome_url_overrides" : {
    "newtab": "my-new-tab.html"
  }
```

In the chrome_url_overrides definition, the value of *newtab* is "my-new-tab.html", which is defined by us. We can define what pages we want.

Our Basic HTML Page

Our next step is to write the HTML page.

```html
<!DOCTYPE html>
  <html>
    <head>
      <meta charset="utf-8"/>
      <title>The best Motivation Tab</title>
    </head>
    <body>
      <h1 id="quote" style="text-align: center"></h1>
      <script type="text/javascript" src="script.js"></script>
    </body>
  </html>
```

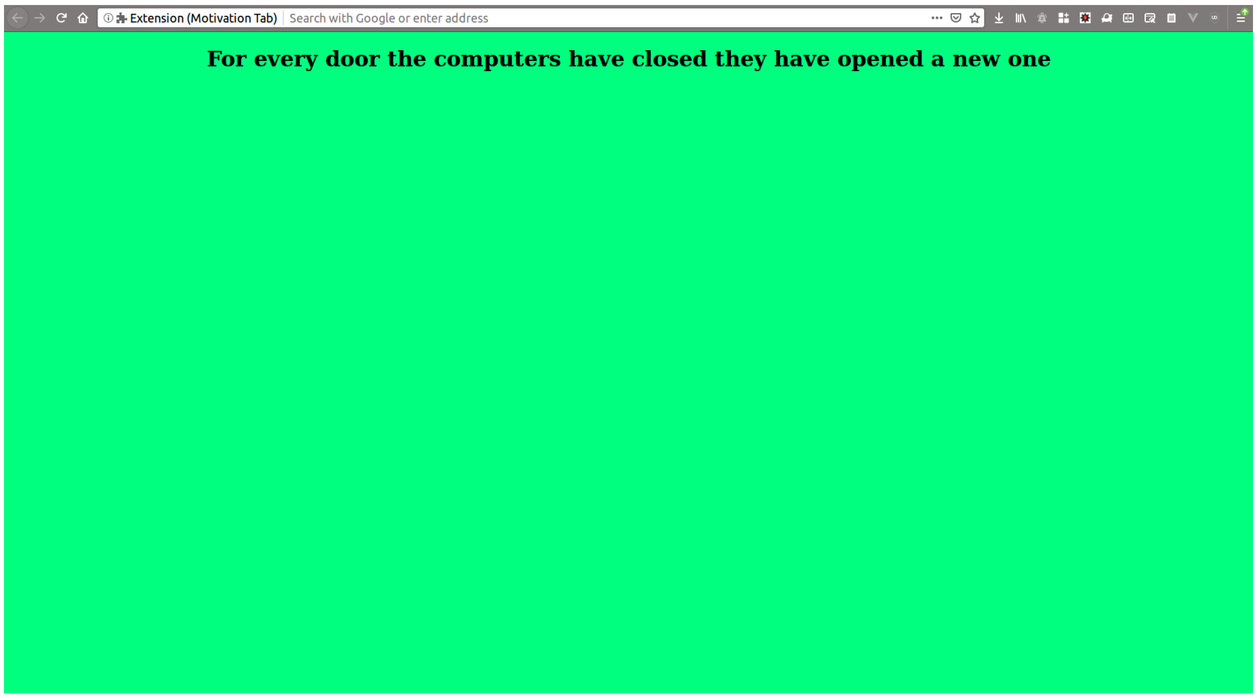
Our HTML page is very minimal, we will be having only **one heading element** with the unique id for that, and we will be including the javascript file.

Our Javascript file,

Our JS file will be having random quotes and the colors for the background.

```javascript
let random_quote = [
 "quote1", "quote2", "quote3"
];

let color_hex = ['#778899','#B0C4DE','#E6E6FA','#90EE90','#00FF7F'];
function getRandomQuote() {
  // Generate Random Number for Quote
  var quote_random_num = Math.floor((Math.random() * random_quote.length));
  // Set the content in the div
  document.getElementById("quote").innerText  = random_quote[quote_random_num];
  // Get Random Number for color
  var random_col_num = Math.floor((Math.random() * color_hex.length));
  // Update the background with color
  document.body.style.backgroundColor = color_hex[random_col_num];

}

// Call the method getRandomQuote when this js loads
getRandomQuote();
```

That's it, our extension (WebExtension or Add-on) is ready, once we do the temporary install we can experience it. When we open a new tab our page by default will look like below.


Extension (Motivation Tab)
Search with Google or enter address

For every door the computers have closed they have opened a new one

New tab Addon

Exercise

Make sure to visit `Mozilla Developer Network (MDN)` [1] to learn more about the above API.

So try out this API in your get started a program. Your task is simple you have to load beautiful images whenever a user opens a new tab. We have already done opening Beautiful Quotes, instead of words you make it an image. Remembers visuals have more power compared to basic text words.

Optional: It will be great if you can share your code or blog about this learning on Twitter. Make sure you use hashtag #WebExtLearn when you are tweeting about this activity.

References

- [1] https://mzl.la/2PBUToi

Experiment 2: Building Power Search Add-on

We got too many browser actions that can be used by developers via WebExtension API(Application Program Interface), we also built our first WebExtension with which we can get motivational images whenever we open a new tab.

Problem Statement and Solution

Problem Statement

When we are surfing a lot there are a lot of cases where we need to search on different search engines. We always don't need to use Google or Bing alone. Sometimes we will directly search in Wikipedia or want to do a video search on Youtube. We are not always active to make everything using Keyboard(KB) because keyboard follow is something like

- select the word using the mouse, right click and copy it or Ctrl+C
- goto the website (if we know the address correctly we will go directly or type in google and go there)
- paste the word which we copied and hit enter.

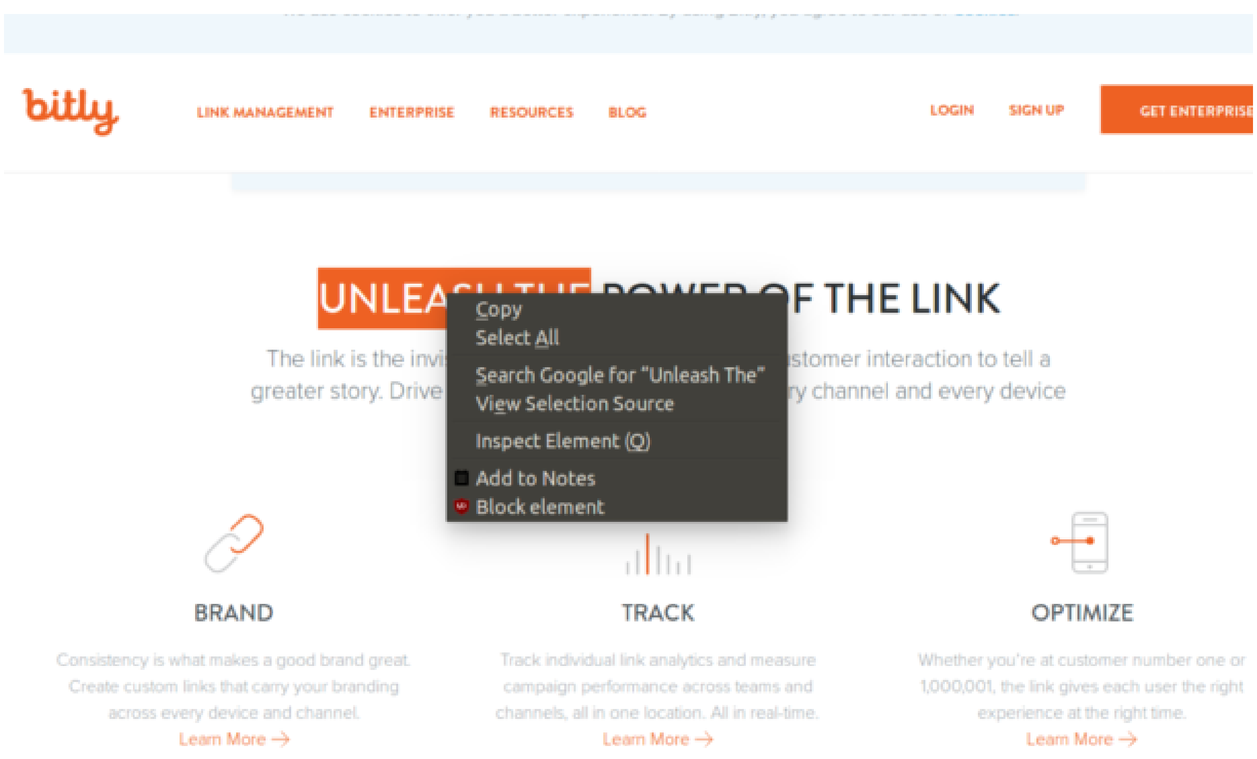
I understand you might have sipped your coffee more than twice here.

Solution

So our good solution will be to use the mouse. The follow with the mouse as follows;

- select the word.
- right click.
- search in our preferred search engine by selecting from the list.

When you are selecting a word and doing the right click, you will be popped up with a list of options. This small window which is coming near your mouse cursor is called as Context Menu.



The link is the invi stomer interaction to tell a greater story. Drive y channel and every device

Consistency is what makes a good brand great. Create custom links that carry your branding across every device and channel.

Track individual link analytics and measure campaign performance across teams and channels, all in one location. All in real-time.

Whether you're at customer number one or 1,000,001, the link gives each user the right experience at the right time.

Context Menu

How we can capture this

Our proposed solution has totally 3 steps. First is selecting the word, once right click is done we have to populate our options (list of search engines available in the browser), then once the preferred search engine is selected then we have to create the new tab. As we have 3 steps here we are going to use 3 simple WebExtension API for this.

- Context Menu API
- Search API
- Tabs.create API (Tabs API has so many sub-API inside them, we will go through them in future)

Since our problem statement is clearly defined and we are going to use this three API only, our WebExtension development will be easier and can be finished quickly.

Building Blocks of Our Power Search Add-on

For this extension, we will learn the API step by step. So at the end joining them will be very easier.

Create the New Tab

To create the tab, we just need the fully qualified URL which we need to open. We need to give URL like “https://www.google.com” instead of “www.google.com”

A sample script is below.

```
browser.tabs.create({
    url:"https://twitter.com/"
  });
```

In the above example, the value of **url** is “https://twitter.com/”, we have to change this if we need to open the different tab.

Context Menu API

Context Menu is nothing but when we click the right click a new pop-up comes near our mouse pointer. We can add our own options in the context menu using Browser Action API. It comes as privileged API, so we need to get **“contextMenus”** permission for using the context menu. In our program, we will be adding the list of search engines in the context.

To add Options in Context Menu

It is very simple to add options in the Context menu.

```
browser.contextMenus.create({
  id: "id1",
  title: "Display Word",
  contexts: ["selection"]
});
```

For each and every context menu item we need to give unique **id** so based on it we can do different actions. In this, the value of **title** is what general users see. In the Previous image **Add to Notes** is the title. Then we need to mention, when we need to show the particular contextMenu item, whether we need to show always or when we select a word (selection) when we select image alone; this is defined in **contexts**

Note: If we have only one context menu for our WebExtension then it will be shown directly if we have more than 1 then under our Extension Name the list will be available.

Getting the selected word

Whenever we click on the contextMenu item we will be getting two things to the listener method we have defined. First one is **info Object** which will have a variety of options based on the context and the Second one is **tab Object** which will have information like tabid, URL and so on. When we want to start the search in our desired website, we need to have the word which is selected, so from the contextMenu item, we will be getting the selected word.

The word which we have click will be available in **info Object** with the key **selectionText**

Search API

Search API will be providing us the details like what are all the search engines installed in the browser, which one is the default, what is their favicon. Search API is also a privileged one, we need to get "**search**" **permission** from the user.

Get list of Search Engine

Search API will be giving the list of search engine installed in our browser. As discussed before we may be having more than 1 search engine in our browser. If it's not available we can install from addons.mozilla.org it will be easier and helpful.

```
var list_of_searchengine = browser.search.get()
```

The above simple snippet will be fetching and giving us the list of search engine available in our machine. Each search engine will be having the values like name i.e., search engine name, may have favIconUrl i.e., search engine favicon URL. We may be using this two things.

Searching in Preferred Engine

Once we got the selectedText from contextMenu our next step would be to search in the preferred search engine. To search in our required engine below code snippet will do.

```
browser.search.search({
    query: "word to search",
    engine: "Wikipedia (en)"
  });
```

In this case,**query** which will have the word to be searched in mandatory, **engine** is not mandatory, if we are not mentioning it, the default one will be taken. In case we mentioned it wrongly exception is thrown.

Assembling our Power Search Extension Parts

Previously we learned about different parts for building our Power Search Extension. We have to create context menu using the search engines available, get selectionText value and search it in the search engine we have.

Writing manifest.json

As usual lets start with writing our manifest file. Here in this time we are additionally introducing three new keys for manifest.json

```
{
"manifest_version": 2,
"name" : "Power Search Engine",
"description" : "Search the selected text in other search engines in your machine",
"version": "1.0",
"icons": {
    "96": "icons/icon-96.png"
  },
  "permissions" : [
          "search","contextMenus"
        ],
"background": {
    "scripts": ["background.js"]
  }
}
```

icons

icons are the JSONObject, which can be used to show the icon for a particular extension. It is good practice to have beautiful icons for our extension. The normal size we should include is 96px X 96px, 60px X 60px.

permissions

Permission is mandatory if we are going to use API with specific data. In this experiment, we are going to use search API and contextMenu API. We have to request permission from the user. Only if the user grants permission we can run this extension.

background

As the name suggests we will have something running in the background. Here we can define the array of scripts so they will be useful and will be listening to everything happens around our extension.

Note: In this program we wont be using any HTML pages, without even HTML page we can run the WebExtension

Our Background Script File

In this experiment we are using only one background script namely background.js; we should have included this in manifest.json else we cant use it.

```
createMenu();

function createMenu() {
// Get the list of search Engine
browser.search.get().then(buildContextMenu);
}
```

So our First step we are going to do in the background is to fetch the search engine and once we are done with fetching we are going to build the context menu.

Whenever the script is loaded, the very first method **createMenu()** is called.

In this method, we have done steps to get the search engine **browser.search.get()**.

.then(some_function_name) represents once the previous process is done successfully jump to the method *some_function_name*

So in our case, once we have got the search engine list using WebExtension API we will be going to the function names **buildContextMenu**

```
function buildContextMenu(searchEngineGot) {

// Iterating Search Engine One by One
for (let searchEngine of searchEngineGot) {

// Create ContextMenu Item
  browser.contextMenus.create({
    id: searchEngine.name, // searchEngine name is set as id.
    title: searchEngine.name,  // searchEngine name is set as title.
    icons: {
      "32": searchEngine.favIconUrl  // searchEngine favicon url is set as icon
    },
    contexts: ["selection"] // context is when we selected a word
  }); // End of contextMenu creation

} // End of iteartion of SearchEngine List

} // End of method
```

Once we got the list of searchEngine successfully we are going to the method named **buildContextMenu**. The parameter which we will get here is the list of SearchEngine Object. This is defined by WebExtension API and we should follow the params as it is.

In this method we are having so many search engines in **searchEngineGot**, so we are iterating one by one.

In one searchEngine we are having **name** and **favIconUrl** value. Additional values are also present but we are not using it. We are then creating the contextMenu with **name** as the *id* and *title*. Then we have to **favIconUrl** which we are using an icon to display the menu item.

```
//When Menu Item is clicked we will goto function menuItemClicked
browser.contextMenus.onClicked.addListener(menuItemClicked);
```

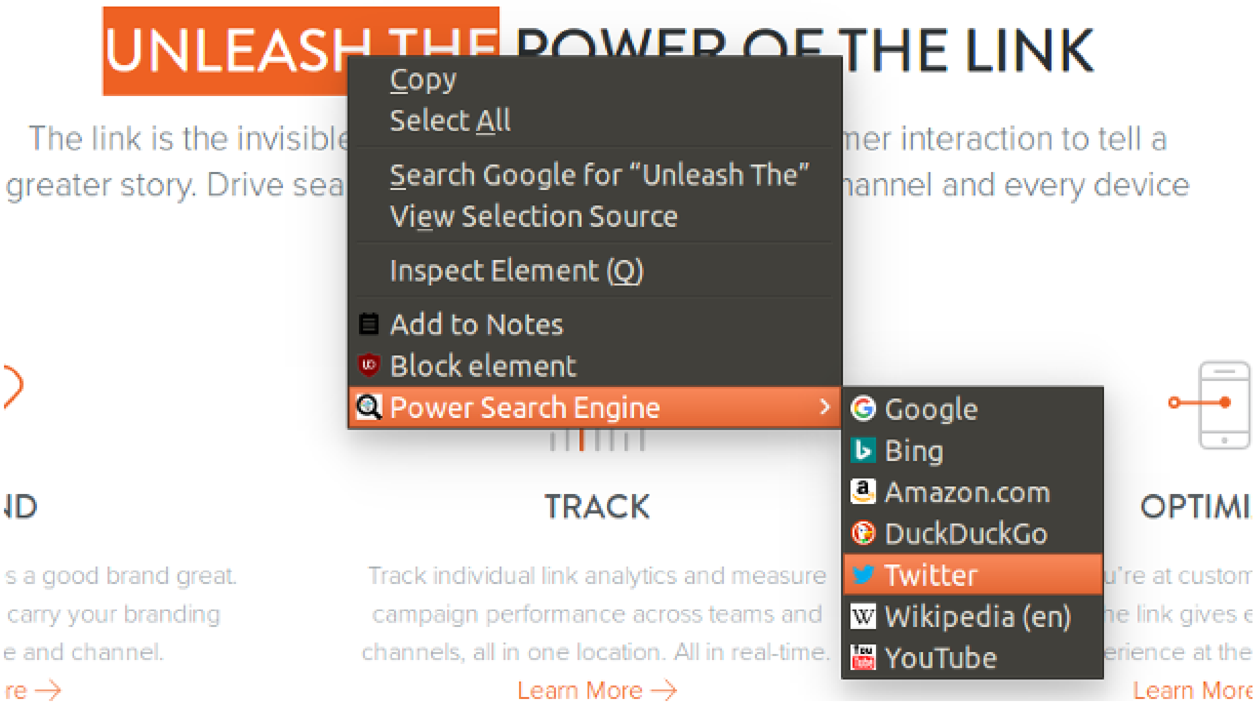
After the menu items are created our next step is to listen whenever a menu item is clicked. This is also provided in WebExtension API. Whenever a menu item is clicked we can have a listener and pass the values to the listener.

```
function menuItemClicked(menu_info_object, tab_object){
// First we have to check whether there is selectionText and also the menuItemId, bo\
th should not be null.
  if(menu_info_object != null && menu_info_object.hasOwnProperty('menuItemId') && me\
nu_info_object.hasOwnProperty('selectionText')){
    browser.search.search({
      query: menu_info_object.selectionText, // Selection text is the text which we \
selected and did right click
      engine: menu_info_object.menuItemId // is the searchEngine.name which we defin\
ed previously.
    });
  }
}
```

We have created a Listener named menuItemClicked; so whenever a menu item is clicked our code flow control will be passed here. We will be getting two objects, one is menu item information object and another one is tab object.

First, we will be checking whether we have the menuItemId and the selected text. Both of them should be not null. And we can also have an additional check whether menuItemId we got matches with the list of search engine we have in our machine.

Once all the preliminary checks are done we will be at our final stage. Now our only step pending is to use the query text and menuitemId(search engine) and search in the respective Website. *browser.search.search* will accept the JSONObject, where *query* key is mandatory here we will be passing the selectedText and for search engine, we have *engine* key.



Power Search Addon Demo

Just in case, if you want to search in the current tab itself instead of opening in the new tab, you have to make slight changes.

- First, in permission, you additionally have to get "**activeTab**" Permission
- Second, you have to make the small modification in the *browser.search.search* API.

```
browser.search.search({
      query: menu_info_object.selectionText, // Selection text is the text which we \
selected and did right click
      engine: menu_info_object.menuItemId, // is the searchEngine.name which we defi\
ned previously.
      tabId: tab_object.id // We will be passing the current tab id here.
    });
```

Exercise

To learn More about `Search API` [1] and `ContextMenu API` [2], visit MDN link given below.

Your simple task here to find out the various search engines (around 15-20). Make them listed in your ContextMenu. This task will involve you to understand the basic how the search Query params work.

For Example: In Wikipedia, if you are search **hello world** you will be getting the URL like below

Wiki URL: https://en.wikipedia.org/w/index.php?search=**hello+world**&title=Special%3ASearch&go=Go

In this, only the highlighted part will be changing every time. Likewise, find for other search engines also and create your extension. You may have to decide what title you have to give, id you have to give.

You may also have to do some little parsing.

Optional: It will be great if you can share your code or blog about this learning on Twitter. Make sure you use hashtag #WebExtLearn when you are tweeting about this activity.

References

- [1] https://mzl.la/2q13lSO
- [2] https://mzl.la/2J3k7tc

Experiment 3: Building Tabs Closer

We have seen previously how to search using different search engine available in our browsers. It should interesting and easy one right? Hope the exercise you tried might have made you find the URL of different search engines and what are all the extra information they are sending as request params.

Problem Statement and Solution

Problem Statement

Now when we are star surfing internet, we end up opening at least 10-20 tabs, because each and every page has so many hyperlinks which we love to learn and come back, but in reality hyperlinks in that pages leads to the different page again there are so many hyperlinks. In such a case if you are in college or office closing the social media websites (some of us do, we don't love to show it to our boss) will be difficult have to close so many tabs after finding them.

Solution

The simple solution will be to have the list of social websites we visit and with one click, we can close all the tabs. Another solution will be to list all the tabs we have close from a pop-up one by one; this will work great when we are having more than 20 tabs.

How we can capture this

Breaking according to two solutions.

Showing the pre-defined Social Network sites

In this type, we will be pre-defining the one social media website which we visit often. We will be having the icon to close that particular social media website alone. On clicking the icon, all the tabs having that social media website URL should be closed.

Fetching all opened URL

Another approach will be fetching all the opened websites, listing them one by one and when we click on the close button we can close them. Just in case we need to visit them we will have another button to visit the particular tab.

In this exercise, we will be learning the following list of API

- tabs.query (To fetch all tabs)
- tabs.remove (Close)
- tabs.move (To display the tab clicked)
- Browser Action Icon
- Browser Action pop-up

Building Blocks of Our Social Media Tab Closer

First, we will take a look at the API we will be using in this Exercise.

Getting started with Basics of API of Social Tab Closer

Starting the Closing Operation

The first step is we need to click on the browserAction icon. As the name suggests browserAction icon will have to scope throughout the browser and it is not specific to any one particular tab. So whenever we click on it irrespective of tabs we can do the procedure. Once we have to click the icon we need to capture the click action.

We have browserAction API for capturing this. *browserAction.onClicked* will be used to listen to click action on that icon.

```
browser.browserAction.onClicked.addListener(action_jump_function_name);
```

Fetching the twitter tabs

In this case, we will be knowing the website URL of Twitter, it is nothing but https://twitter.com/. So our first step will be to get all the tabs where Twitter is loaded. Our WebExtension `tabs.query` have a lot of combination to get(or fetch) the tabs loaded, in this case, we are going to fetch based on **URL**

```
browser.tabs.query({url: "*://*.twitter.com/*"});
```

Sometimes we won't know whether its *HTTP* or *HTTPS* so we are using the regex (Regular Expression) *://* before twitter.com and also we may be having anything loaded so we have /* at the end of .com

Close the tabs

Once we have got the tabs (or the single tab) we need to close them. We have to get tabid from the tab object we got, based on that only we will be closing the tabs.

```
var removing = browser.tabs.remove([tab_id1, tab_id10, tab_id15]);
```

Assembling parts of our Extension

So now our task is to just write all the above code in manifest.json and background script file. Mos to the things is covered.

Writing manifest.json

```
{
  "manifest_version": 2,
  "name" : "Social Tab Closer",
  "description" : "Close the pre-defined Social Media Tabs",
    "version": "1.0",
  "icons": {
      "96": "icons/icon-96.png"
    },
    "permissions" : [
      "tabs"
    ],
  "background": {
      "scripts": ["background.js"]
    },
    "browser_action": {
      "default_icon": {
        "96": "icons/icon-96.png"
      }
    }
  }
```

Here changes we made will be getting the tabs permission. And also we have to define the brower_-action icon alone. We won't make any other drastic changes compared to the previous manifest.json.

Writing our background.js

We will be splitting the background javascript into parts.

- Listen to browser Action Icon
- Fetch the twitter tabs.
- close the tabs.

Listen to browser Action Icon

```
browser.browserAction.onClicked.addListener(icon_clicked);
```

Our first step will be to listen whenever our browserAction icon is clicked. Once it is clicked we have to pass the control to the method (icon_clicked) which we have defined.

Fetch the twitter tabs

```
function icon_clicked(tab){
  var querying_tabs = browser.tabs.query({url: "*://*.twitter.com/*"});
  querying_tabs.then(close_tabs);
}
```

Our next step will be to fetch(query) the tabs where twitter.com is loaded, so we will be using the regex which we discussed before and fetch the tabs with the URL pattern.

Close the tabs

```
function close_tabs(twitter_tabs){
var array_of_twitter_tabid = [];
  for(let single_twitter_tab of twitter_tabs){
    array_of_twitter_tabid.push(single_twitter_tab.id);
  }

  browser.tabs.remove(array_of_twitter_tabid);
}
```

Our Next step is to get the tab id with the tabs we have fetched. We are first iterating each and every tab object which we are getting in tab array as a result of the query using URL. Then while iterating we are getting the **id** and inserting in the array. After we finish the iteration we are just using the close (remove) API and closing the tabs.

Building Blocks of All Tabs Closer

We are going to almost use the same set of API which we used for our social tab close. Here we will additionally have a very small change in manifest.json.

Getting started with Basics of API of All Tab Closer

Adding popup html page

Previously we have mentioned the browserAction icon and just left. Now in manifest.json, we are additionally going to add the HTML page which has to be shown when a browserAction icon is clicked and also the title which has to be shown when we move our mouse above browserAction icon.

Note:We can either Click on the browser Action icon or we can show the HTML page

Fetching the all the tabs

Previously we have passed the URL and fetched the tabs where twitter.com is loaded. Now simply we don’t need to pass anything so we will be getting all the tabs.

```
browser.tabs.query();
```

Close the single tab

We are going to close the tabs one by one, so instead of passing the array of tabID's we can pass only one tabID also.

```
var removing = browser.tabs.remove(tab_id1);
```

Assembling parts of our All Closer Extension

Let's start with manifest.json, for now, will share the whole content, in future, we can slowly start sharing only the parts which are having the difference.

manifest.json look

```
{
  "manifest_version": 2,
  "name" : "All Tab Closer",
  "description" : "Close All Tabs",
    "version": "1.0",
  "icons": {
      "96": "icons/icon-96.png"
    },
    "permissions" : [
    "tabs"
    ],
  "background": {
      "scripts": ["popup/script.js"]
    },
    "browser_action": {
      "default_icon": {
        "96": "icons/icon-96.png"
      },
      "default_title": "List all tabs",
    "default_popup": "popup/tabs_list.html"
    }
  }
```

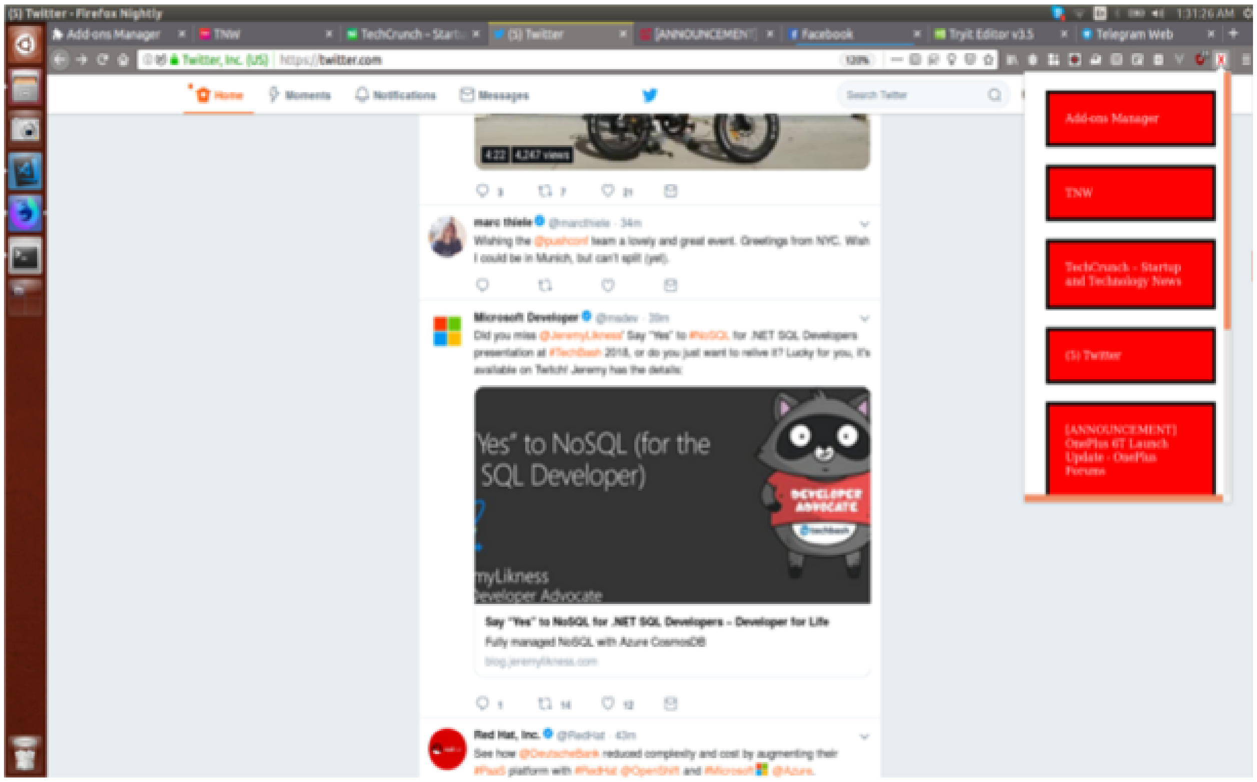
So our change will be mostly in the **browser_action** part. We have additionally added two more keys for it. Most of the parts remain quite familiar and its based on this Add-on.

Building our tiny HTML page.

We need an HTML page which will be acting as a popup when we are going to click on **browserAction** icon. So we need to design a very small page.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8"/>
    <title>List of Tabs</title>
</head>
<style>
    .single_div {
        background-color: lightgrey;
        border: 5px solid rgb(11, 12, 11);
        padding: 25px;
        margin: 25px;
    }
    </style>
<body>
  <div id="tabs_list" style="width: 300px">
    </div>
  <script type="text/javascript" src="script.js"></script>
</body>
</html>
```

Our HTML page is having some CSS for design and most important part is the div with id *tabs_list*, we will be creating all our tab details and will show here.


Add-ons Manager
TNW
TechCrunch - Startup and Technology News
(5) Twitter
[ANNOUNCEMENT] OnePlus 6T Launch Update - OnePlus Forums

Popup Before Closing any tab

And we have also included the Javascript file at the end. This same JS file will be acting as a background script file.

Included JS and Background Script File

Our first action once the javascript file is loaded is to collect all the tabs which are opened. We are processing this action in **startFetchingTabs** method

```
startFetchingTabs();

// Fetch all the tabs
function startFetchingTabs() {
  var querying = browser.tabs.query({});
  querying.then(create_list_tabs, onError);
}
```

After we have collected the tabs our action will be to list them in the popup browserAction page. So we are iterating tab object one by one. tab object will be having details like tabID, tabTitle, and tabURL. We as of now need tabID and tabTitle.

First, we will be deleted all the already listed tabs in the pop-up HTML page, then after that, we are creating the div one by one for all the tabs opened.

```
function create_list_tabs(fetched_tabs) {
  // delete already existing child nodes.
  if (document.getElementById("tabs_list").childNodes.length > 0) {
    var myNode = document.getElementById("tabs_list");
    while (myNode.firstChild) {
      myNode.removeChild(myNode.firstChild);
    }
  }

  // Create new child nodes.
  for (let single_tab of fetched_tabs) {

    var created_div = document.createElement("div");
    created_div.style.width = "200px";
    created_div.style.background = "red";
    created_div.style.color = "white";
    created_div.innerHTML = single_tab.title;
    created_div.id = single_tab.id;
    created_div.className = "single_div";
    created_div.addEventListener("click", function () {
      close_tab(single_tab.id);
    }, false);

    document.getElementById("tabs_list").appendChild(created_div);
  }
}
```

We have created the div in such a manner, the tab Title will be displayed to the user. And each and every title is associated with tabID. When we click on any part of the div, the tab ID is passed to **close_tab** method and there we will start the process of closing the tab.

```
// Close the tab and recreate the popup html
function close_tab(tab_id_to_close) {
  browser.tabs.remove(tab_id_to_close);
  startFetchingTabs();
}
```

As discussed before, once the title is clicked, the respective id is passed and we will be closing the tab with that particular ID after that we will again be creating the popup browserAction HTML page.

Exercise

Make sure to visit Mozilla Developer Network (MDN) to learn more about the above API.

- `Close API` [1]
- `Query API` [2]
- `Capture Tab` [3]
- `Reload Tab` [4]
- `Browser Action` [5]

There are a lot of things related to tabs API to be explored. We have as of now have used the title and showed the list of tabs. But sometimes Title alone won't be enough, we have an API to capture the current instance of the tab as an image, use this API instead of the title make sure to show the image and on the image have two options one for reloading and another for close. So based on the button clicks use Reload API or Remove API.

Optional: It will be great if you can share your code or blog about this learning on Twitter. Make sure you use hashtag #WebExtLearn when you are tweeting about this activity.

References

- [1] https://mzl.la/2QZ1PfB
- [2] https://mzl.la/2PfiZYW
- [3] https://mzl.la/2yLRH2g
- [4] https://mzl.la/2yrb5SZ
- [5] https://mzl.la/2OCdEf1

Experiment 4: Building PDF WebPage Extension

So we were learning about fetching the tabs Closing them one by one. There are times in our life where we need to do something later. Yes, we need to do a lot of things later. In this chapter we will be learning how to save so we can check it later. We learned more about browserAction icon previously now we will be learning opposite to it (not exactly just for fun) PageAction icon, we are seeing it in a lot of pages think of closing tab button, bookmark button, what they are doing ? what is their scope?

Problem Statement and Solution

Problem Statement

We read each and everyday lot of articles. Think of pages you read in Wikipedia, sometimes you will dive deep into it and wanted to spend the large amount on that article but you might have closed it and never opened again. This is the saddest thing can happen when you see that article again and start reading.

Solution

We have very simple solutions for this, we can save it to the Pocket app (https://getpocket.com/), or you can save the whole page by selecting all the text copying it and saving it in the text file which is the seriously boring thing you can do. One better option is to make the pages as PDF when we have a readable view.

How we can capture this

We have very rich tabs WebExtension API which with which we can know whether that particular tab is in readable mode or not, we can know when there is a change in the tab, with the simple click we can make the page as PDF.

Building Blocks of PDF WebPage Extension

We will get straight to the API we will be using. Our flow is simple, whenever there is a change in tab we will be checking whether current tab has a readable formatted page, if it is available we will be enabling the pageAction icon, whenever there is a click in pageAction icon we will be saving it as PDF in our system.

Capturing change in tab

Changing the tab means activating from one tab to another tab. In Tabs API we are having the method to capture this.

```
  browser.tabs.onActivated.addListener(listen_to_tab_change);

  function listen_to_tab_change(activeTabInfo){
    //do Something
  }
```

So whenever there is a change in the tab, if we want to capture it we will be having an activeListener method. In this method, we will be doing whatever we want.

Similarly, we have to check whenever there is an update on the same tab(current tab itself). Like if the current page is www.google.com and now we moves to Wikipedia site where we may some downloaded content, so we have to capture that also.

```
browser.tabs.onUpdated.addListener(listen_to_tab_update);
 function listen_to_tab_update(activeTabInfo){
    //do Something
  }
```

Finally we have to listen when there is a new tab create.

```
browser.tabs.onCreated.addListener(listen_to_tab_create);
function listen_to_tab_create(activeTabInfo){
    //do Something
  }
```

These above 3 are the cases we need to capture. In all the 3 cases the following will be the same. We will first check whether the tab updated or created or changed to is in ReadMode format.

Checking is current Tab reabable format

So the tab object will be having a lot of details like id, index, title, URL and so on. One of the value is **isArticle**

If the value of isArticle is true, then it means we have readable format (ReadMode is available) tab. In that case, we can download as PDF.

Showing the pageAction icon

So we came to know the current page can be in reading mode, which means we can have the option to download the page as PDF. At this stage, we may need to have user intervention. Remember we learned about browser Action icon during **Building our Tabs Closer**, similar we have pageAction icon (which we can consider limited edition of browser Action icon).

The major difference between the browserAction icon and pageAction icon is the scope of availability. the browserAction icon is available outside the address bar icon which means it is not limited to that particular tab alone while the pageAction icon is inside the address bar which means it is very specifically limited only to that tab alone.

So in our case, we will be showing the pageAction icon whenever the current page which we are viewing is having a Reader mode.

Downloading the page

The only final step now is we have to download (save) the page as PDF. This is just one method call, all parameters are optional so we need not pass anything to just save without any modification.

```
browser.tabs.saveAsPDF({});
```

So our next step is to start assembling all the API's into single WebExtension.

Assembling parts of PDF-IT Extension

manifest.json look

Most parts of our manifest.json look very similar. The only addition to their permission is **activeTab**. And we are adding a new part of json **page_action** which is similar to browser action.

```
{
"manifest_version": 2,
"name" :"Save as PDF",
"description" : "Saves pages as pdf",

"homepage_url": "http://iamvp7.github.io/",
  "version": "1.0",

"icons": {
    "48": "icons/page-48.png",
    "32": "icons/page-32.png"
  },
  "background": {
    "scripts": ["background.js"]
  },
  "permissions" : ["tabs" , "activeTab"],
  "page_action": {
    "default_icon": "icons/page-32.png",
    "default_title": "Download as PDF"
  }
}
```

Adding tab state changes listeners

Our next step is adding the listeners when there is an update in the tab, creation in a new tab, activation in the tab.

```
browser.tabs.onUpdated.addListener(listen_tab_update);

browser.tabs.onActivated.addListener(listen_when_tab_activated);

browser.tabs.onCreated.addListener(listen_when_tab_create);
```

After listen we will have the same set of actions, but different listeners have the different structure of call back functions. So we need 3 methods.

Adding Implementation for listeners

So we will be adding the Listener for each and every type we have mentioned. In some cases we will be getting the current tab id directly, in some cases, we will be fetching them.

Adding onUpdate Listener

When there is an update in the current tab we will be listening to the change in this method for this extension.

```
function listen_tab_update(tabId, changeInfo, tab) {
  getCurrentTabDetails();
  }
function getCurrentTabDetails(){
    var querying = browser.tabs.query({currentWindow: true, active: true});
    querying.then(getInfoForTab, onError);
  }

  function getInfoForTab(tabs) {
    if (tabs.length > 0 && tabs[0] != null && tabs[0].isArticle) {
      browser.pageAction.show(tabs[0].id);
    }
  }

  function onError(error) {
    console.log(`Error: ${error}`);
  }
```

The callback in this function will be having 3 parameters tabId in which the update action has taken place, change info option and the current state of the tab object. Among the all 3 we have got we will be needing the **tabId** alone, so for the current tab we will be showing the **pageAction icon**.

But we also need the details like whether the current tab is in a readable format. So first in **getCurrentTabDetails** we will be querying the active tab from the currently active window, we will be getting the list of tab array. From that tab array, we will iterate each object and will be getting the first one alone.

Adding onActivated Listener

So we have to add a listener method for tab activated. In this, we will be receiving only one parameter (activeInfo Object), where we won't get the tab details. So for this, we will be querying the current tab and from that, we will be receiving the tabId. Similar to what we have done during close tabs we will do here.

```
  function listen_when_tab_activated (activeInfo) {
    getCurrentTabDetails();
  }
```

Here also similarly we will be call the **getCurrentTabDetails**

Adding onCreated Listener

So our final listener method is required when we are creating the new tab.

```
function listen_when_tab_create(tab){
  getCurrentTabDetails();
}
```

Here also similarly we will be called the **getCurrentTabDetails**

The only thing which we are doing in each of the listener methods is to show the **pageAction** icon. As discussed previously, when the user clicks we will be saving it as PDF

Adding pageAction listener

So when the pageAction icon is clicked, we have to do something. So we will be listening from backgroun.js and will wait for pageAction click.

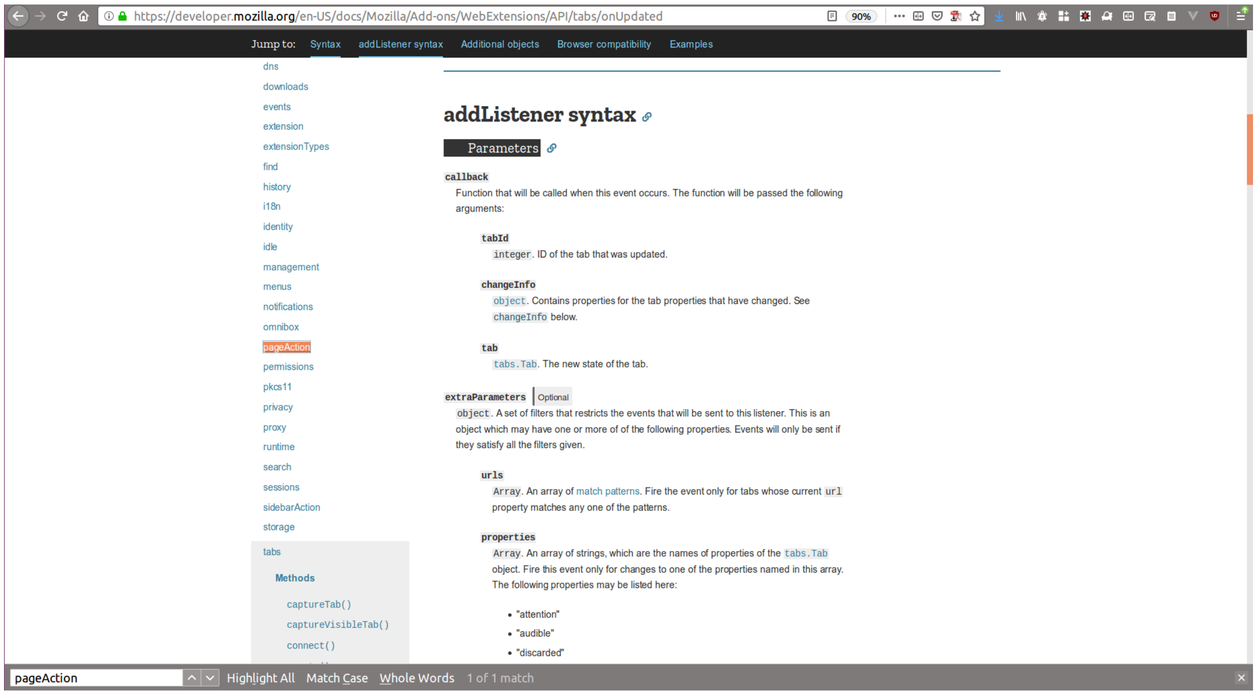
```
browser.pageAction.onClicked.addListener(doPageActClickProc);
```

Save as PDF

So our final step once the pageAction icon click is just to save the current readable page as PDF.

```
function doPageActClickProc(){
  // browser.tabs.toggleReaderMode();
  browser.tabs.saveAsPDF({});
}
```

So when you are running your extension you will be seeing the icon near the bookmark icon inside the address bar. Like below


https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/tabs/onUpdated

Jump to: Syntax addListener syntax Additional objects Browser compatibility Examples

dns
downloads
events
extension
extensionTypes
find
history
i18n
identity
idle
management
menus
notifications
omnibox
pageAction
permissions
pkcs11
privacy
proxy
runtime
search
sessions
sidebarAction
storage
tabs
Methods
captureTab()
captureVisibleTab()
connect()

addListener syntax

Parameters

`callback`
Function that will be called when this event occurs. The function will be passed the following arguments:

`tabId`
`integer`. ID of the tab that was updated.

`changeInfo`
`object`. Contains properties for the tab properties that have changed. See `changeInfo` below.

`tab`
`tabs.Tab`. The new state of the tab.

`extraParameters` Optional
`object`. A set of filters that restricts the events that will be sent to this listener. This is an object which may have one or more of of the following properties. Events will only be sent if they satisfy all the filters given.

`urls`
`Array`. An array of match patterns. Fire the event only for tabs whose current `url` property matches any one of the patterns.

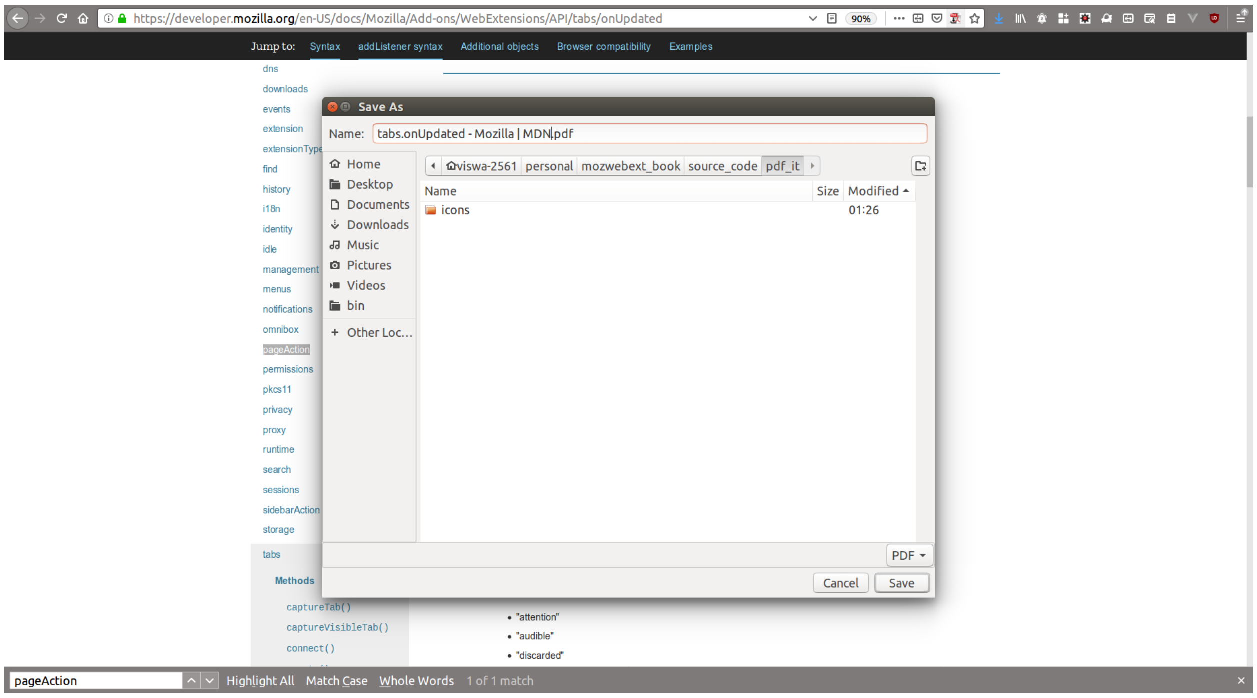
`properties`
`Array`. An array of strings, which are the names of properties of the `tabs.Tab` object. Fire this event only for changes to one of the properties named in this array. The following properties may be listed here:

- "attention"
- "audible"
- "discarded"

pageAction Highlight All Match Case Whole Words 1 of 1 match

PDF IT PageAction Icon

When we click on the icon we will be getting the pop-up for saving the file as pdf.


https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/tabs/onUpdated

Save As

Name: tabs.onUpdated - Mozilla | MDN.pdf

viswa-2561 personal mozwebext_book source_code pdf_it

Home
Desktop
Documents
Downloads
Music
Pictures
Videos
bin
Other Loc…

Name	Size	Modified
icons		01:26

PDF

Cancel Save

pageAction Highlight All Match Case Whole Words 1 of 1 match

PDF IT Save file

Exercise

To learn More about below list of API, visit MDN link given below.

- `onUpdate` [1]
- `onActivate` [2]
- `onCreate` [3]
- `pageAction` [4]

Your simple task is to create a small social share button. So when the user clicks on the pageAction icon he should have options to share it to various social media.

Optional: It will be great if you can share your code or blog about this learning on Twitter. Make sure you use hashtag #WebExtLearn when you are tweeting about this activity.

References

- [1] https://mzl.la/2OngPCk
- [2] https://mzl.la/2JDMsqq
- [3] https://mzl.la/2EYAnx9
- [4] https://mzl.la/2SHe02h

Experiment 5: Building Water Notifier Extension

Humans depend on a lot on the notification system. The simplest thing and more like a hello world program for WebExtension are building Notification system. I got started with sending the simple notification.

Problem Statement and Solution

Problem Statement

We are in a very busy life where we even forget to drink the recommended about of water for our body. This leads to dehydration and we start performing badly. One of the simplest solutions is being well disciplined and drinking 1 glass of water for every 45 mins, this won't happen unless we have self-control. Another solution is having an alarm in mobiles, sometimes if it rings people around us will get irritated. Another solution which can be done with WebExtension is sending us notification in our browser for every 45 mins so we know some actions to be done.

Solution

So our solution is simple, have an alarm in our system so we will be sending the notification every 45 mins.

How we can capture this

Our proposed system has clearly mentioned we need to have an alarm in our browser, and to show something in UI for that let's have a small Notification UI. We are having the following list of WebExtension API.

- `alarms API` [1]
- `notifications API` [2]
- `Runtime API` [3]

In our system we need to set alarm for every 45 mins from the time we started our browser, so we need to find out when the browser is started so for that we need the help of **Runtime API**. This Extension is one of my favorites as I started learning WebExtension development with this.

Building Blocks of Our Water Notifier Extension

We can develop this extension very easily. Our building blocks for this is very simple and we will use simple 3 API for the whole process.

Finding the Browser Startup & Add-ons Installation

Our first step is to find out when the Add-ons is installed or when the Add-on is opened. From that time we have to calculate 45 mins and send the Notification.

```
// Add a method what to happen when Addon is installed
browser.runtime.onInstalled.addListener(handleInstalled);

// Add a method what to happen when browser starts
browser.runtime.onStartup.addListener(handleStartup);
```

We have two different Listener. First one is when the Extension is installed and the second one is when we are starting our Browser.

Setting the Alarm

Our next step after finding the Add-ons installation or browser startup is to set the alarm.

```
browser.alarms.create("my-periodic-alarm-install", {
  when,
  periodInMinutes
});
```

Our good practice is to give first the name of the alarm. In our above example **my-periodic-alarm-install** is the name of the alarm.

After that, we will be mentioning the value of **when**, the time at what the very first alarm should be triggered. Secondly, we will also be giving the value of **periodInMinutes** this is set to 45 because we wanted to trigger the alarm every 45 mins.

If we already have an alarm with the same name for this extension, then the previous alarm will be deleted and the new alarm will be set.

Capture the Alarm

Once we set the alarm our next step is to listen in the background when our alarm is triggered and we have to capture it and do our processing. Even in our mobiles, when the alarm signal is the trigger it is captured then some sounds (alarm tone) is made to play, we should understand when the alarm is triggered it means signals are triggered not the sound.

```
  browser.alarms.onAlarm.addListener(handleAlarm);
```

We have a listener method in alarms API for this. So we will have the method and do the actions we need to do.

Fire the Notification

After we have captured the alarm triggered our final step is to show the notification.

```
browser.notifications.create(waterNotification, {
    "type": "basic",
    "iconUrl": browser.extension.getURL("icons/icon-96.png"),
    "title": "Time for Water",
    "message": "Drink enough water to stay healthy"
  });
```

We will have the notification ID at the place of **waterNotification**. As of now, Firefox is supporting only the basic type of notification. We can have a small **title** text and a descriptive **message** alone in our notification.

Note in case we call notification creation more than once in rapid succession, Firefox may end up not displaying any notification at all.

Assembling our Water Notifier Extension Parts

So almost everything what we have discussed is available as a single code file (background.js). If we copy and paste with one addition of method that's enough to make this Extension work.

Writing manifest.json

Our manifest.json is very simple. The only change which is available in at the permission's part. We need alarms permission and notification permission.

```
{
"manifest_version": 2,
"name" : "Notification Alarm",
"description" : "Sending Notification and Alarm",
"homepage_url": "http://iamvp7.github.io/",

"version": "2.0",
"permissions" : [
  "notifications",
  "alarms"
  ],
"background": {
    "scripts": ["background.js"]
  }
}
```

Adding listeners

So as discussed before we are having two listeners in our program.

```
// Add a method what to happen when Addon is installed
browser.runtime.onInstalled.addListener(handleInstalled);

// Add a method what to happen when browser starts
browser.runtime.onStartup.addListener(handleStartup);
```

Adding Implementation for listeners

Both the listeners have different parameters in the callback function. So we have to define two methods.

Listener for startup

```
// Method handling when the browser starts
function handleStartup() {
  createAlarm();
}
```

So this method won't have any params. The only action we are doing from here is to move the call where we can create the alarm.

Listener for installation

```
// Method to handle when the addon is installed
function handleInstalled(details) {
  createAlarm();
}
```

This callback method will have installation details like ID, previous version. Here also we are just moving to create the alarm method, no other action is done.

Create the alarm

So after listening to installation and startup our step is to create the alarm.

```
function createAlarm(){
  const when = nearestMin30();
  browser.alarms.create("my-periodic-alarm-install", {
    when,
    periodInMinutes
  });

}

// Finding the nearest 30 Mins
function nearestMin30(){
  var now = new Date();
  now.setMinutes(d.getMinutes() + 45);
  return  now.getTime();
}
```

We will be finding the nearest 45 mins from startup/installation and we will be setting the alarm. The method **nearestMin30** job is to find the nearest 45 mins and return its milliseconds value.

Adding listeners for alarms

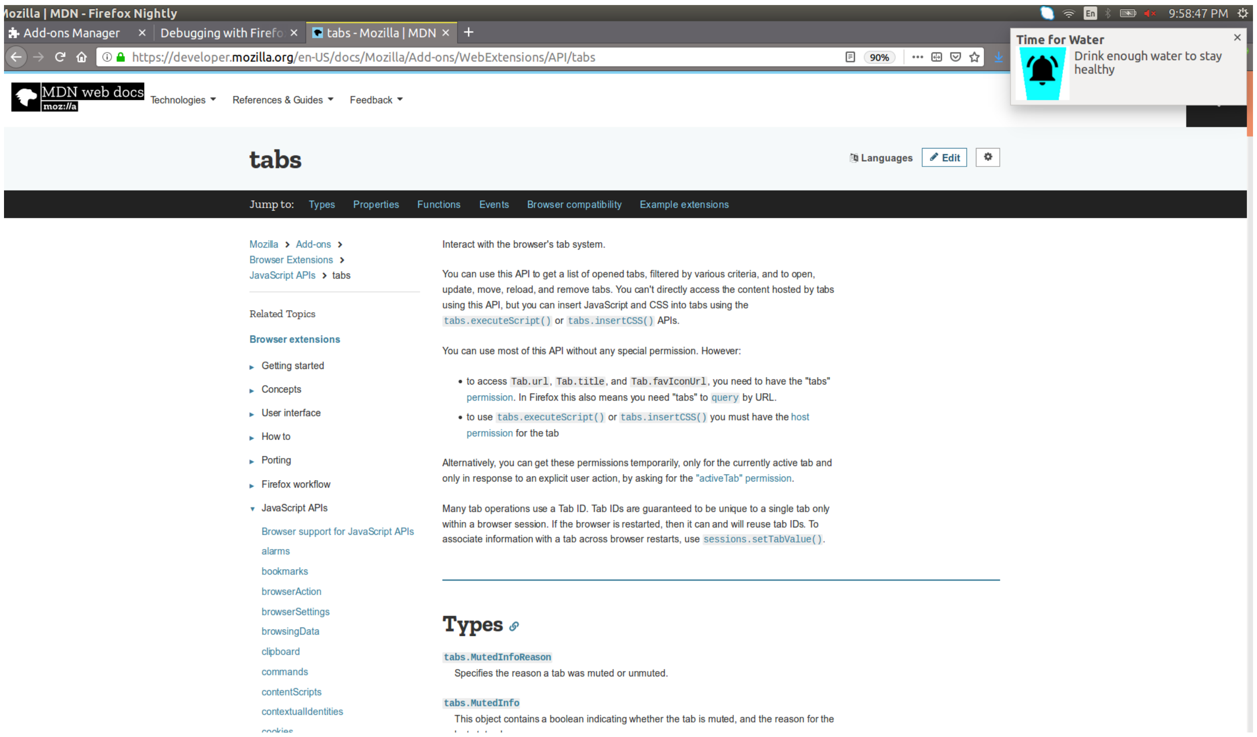
Our major one line of this program is adding a listener for the Alarms. Once the alarm is set we should be listening for its trigger in the background always. If the trigger is set we need to do our required action, in our case it's pushing the notification.

```
browser.alarms.onAlarm.addListener(handleAlarm);
```

Push the notification

So our final step is to push the simple notification. A simple notification will come, so we can understand water has to drink at that time.

Before installing the extension for testing bring the time to 1 min or 2 mins. After installing the Extension, wait for the mentioned time and a notification will be coming in soon.



Time for Water

Drink enough water to stay healthy

tabs

Interact with the browser's tab system.

You can use this API to get a list of opened tabs, filtered by various criteria, and to open, update, move, reload, and remove tabs. You can't directly access the content hosted by tabs using this API, but you can insert JavaScript and CSS into tabs using the `tabs.executeScript()` or `tabs.insertCSS()` APIs.

You can use most of this API without any special permission. However:

- to access `Tab.url`, `Tab.title`, and `Tab.favIconUrl`, you need to have the "tabs" permission. In Firefox this also means you need "tabs" to `query` by URL.
- to use `tabs.executeScript()` or `tabs.insertCSS()` you must have the host permission for the tab

Alternatively, you can get these permissions temporarily, only for the currently active tab and only in response to an explicit user action, by asking for the "activeTab" permission.

Many tab operations use a Tab ID. Tab IDs are guaranteed to be unique to a single tab only within a browser session. If the browser is restarted, then it can and will reuse tab IDs. To associate information with a tab across browser restarts, use `sessions.setTabValue()`.

Types

`tabs.MutedInfoReason`
Specifies the reason a tab was muted or unmuted.

`tabs.MutedInfo`
This object contains a boolean indicating whether the tab is muted, and the reason for the

Water Notification

Exercise

To learn More about below list of API, visit MDN link given below.

- `alarms API` [1]
- `notifications API` [2]
- `Runtime API` [3]

Your simple task is to create a WebExtension which will have a pageAction icon, on click it we should make the article to be opened after 30 mins. This Extension will look like a simple version of Firefox Test pilot Snooze tab. In this, we will be using Open tab API, Alarms API, pageAction API.

Optional: It will be great if you can share your code or blog about this learning on Twitter. Make sure you use hashtag #WebExtLearn when you are tweeting about this activity.

References

- [1] https://mzl.la/2zBlyuT
- [2] https://mzl.la/2zrMn4r
- [3] https://mzl.la/2zqzxDE

Experiment 6: Building Shortcuts for Youtube Controller

Problem Statement and Solution

Problem Statement

Think of the busy working day. To relax you should be playing some of your favorite songs on Youtube. And as you continue listening you will start browsing the internet. It is very easy to open so many tabs and it will become difficult to find the Youtube tab which you opened long back. You may love to skip the song when some irritating song comes or you would like to switch between pause and play when someone comes and talks to you.

Solution

In such a case we should be quickly able to find the Youtube tab and do the action. There are two ways we can do this.

The first solution is to make a small pop-up with different buttons like play/pause and next song.

The second solution is to make use of the shortcuts. Yes, we can build our own shortcuts in Firefox.

In this tutorial, we will learn how to creat our own shortcuts in an extension.

How we can capture this

As our proposed solution is related to shortcuts we have a WebExtension API named `commands` [1] which we can use to build our own shortcuts for Firefox. And after creating shortcuts we can programmatically click the next song button or play/pause button on Youtube. To achieve this we need to write some minimal script to click the buttons. After we write the script we have to execute them, to this in the Tabs API we have a special method called **executeScript()** [2], similarly, we also have for CSS named **insertCSS()** which we won't use as of now.

Building Blocks of Our Youtube Controller Extension

Define the Shortcut

Our very first task will be to define the shortcut which we are going to use for our actions. We need to define unique shortcuts for both the set of actions. Our shortcuts will be defined in the manifest.json

```
"commands": {
  "next-song": {
      "suggested_key": { "default": "Ctrl+Alt+N" },
      "description": "Go to next song"
    }
}
```

Each and every command will have a name, in our example **next-song** is the name of the shortcut. Each command will have two properties *suggested_key* and *description*. **Description** is the just one line about this particular shortcut. And **suggested_key** will have the set of shortcuts based on the operating system. List of keys which can be present in the suggested_key json is shared below. **default** is not specific to any platform it will be used in all platforms.

```
"default", "mac", "linux", "windows", "chromeos", "android", "ios"
```

Listening to commands

In our background script, we should be listening when our command is triggered. We have **onCommand.addListener** method for this. In this, for the call back function, we will be receiving the **command_name** as the argument. Based on the **command_name** we have to execute different scripts.

```
  browser.commands.onCommand.addListener(on_command_received);

  function on_command_received(command_name){
    // Do something based on command_name
  }
```

Execute the Click Scripts

So we have manually found out the button ID in the Youtube. It should be done before starting the development of extension. It is one time process only.

```
// Script for clicking the next button
document.querySelector(".ytp-next-button.ytp-button").click();

// Script for clicking the play/pause button
document.querySelector(".ytp-play-button.ytp-button").click()
```

The first part of executing the script is to find in which tab the *Youtube* is running. To this, we have already using `tabs.query` API. We just need to change the value of the variable **url**

```
  var querying = browser.tabs.query({
    url: "*://*.youtube.com/*"
  });
```

So from this, we will be getting the tabs object, in that for the first tab we will be getting the ID and executing the script on that tab alone.

```
  var executing = browser.tabs.executeScript(
    tabID, {
      code: commandToExecute
      });
```

Here the value of **commandToExecute** is the script which we have got earlier for programatically click the play/pause or next song button.

Assembling our Power Search Extension Parts

manifest.json file

As usual, we will be starting with our manifest.json file and will make the single script file.

```
{
    "manifest_version": 2,
    "name": "Youtube controller Using Shortcuts",
    "description": "Control Youtube play without visiting the tab.",
    "homepage_url": "http://iamvp7.github.io/",
    "version": "1.0",
    "icons": {
        "96": "icons/icon-96.png"
    },
    "background": {
        "scripts": ["background.js"]
    },
    "permissions": [
        "*://*.youtube.com/*",
        "tabs"
    ],
    "commands": {
        "next-song": {
            "suggested_key": {
                "default": "Ctrl+Alt+N"
            },
            "description": "Go to next song"
        },
        "pp-song": {
            "suggested_key": {
                "default": "Ctrl+Alt+P"
            },
            "description": "play or pause the song"
        }

    }
}
```

In this manifest.json in permission, we will be requiring the website permission of youtube and tabs because we are going to execute the script where youtube in running.

And the newest part of manifest.json you will notice is the **commands** object. We will be defining what are the commands we will be using and what are their shortcuts and their description in manifest.json itself.

Our Background Script File

```
var commandToExecute = '';

browser.commands.onCommand.addListener(on_command_received);
function on_command_received(command_name) {

    if (command_name == 'next-song') {
        commandToExecute = 'document.querySelector(".ytp-next-button.ytp-button").cl\
ick()';
    } else if (command_name == 'pp-song') {
        commandToExecute = 'document.querySelector(".ytp-play-button.ytp-button").cl\
ick()';
    }

    // Query tabs based on url
    var querying = browser.tabs.query({
        url: "*://*.youtube.com/*"
    });
    querying.then(executeOnTab, onError);
}


```

```
function executeOnTab(tabs) {
    var tabID = tabs[0].id; // getting the first tab ID alone

    var executing = browser.tabs.executeScript(
        tabID, {
            code: commandToExecute
        }
    );
    executing.then(onExecuted, onError);
}

function onError(error) {
    console.log(`Error: ${error}`);
}


function onExecuted(result) {
    console.log(`We executed script in the first youtube tab`);
}
```

So our background script file is well explained and is organized well.

First, we will have a listener (**onCommand.addListener**) to listen when the shortcut is triggered. We have a callback function (**on_command_received**) for this listener. This callback function will be receiving the command as the argument.

Then based on the command (either *next-song* or *pp-song*) we will be assigning the script which we need to execute. After that our process is to find the first Youtube tab. Using the **tabs.query** the method we will be getting the **tabId** of first Youtube tab.

As we have the required tab ID and the script to execute, we will be using **browser.tabs.executeScript** method to execute the script and make our necessary changes.

Note: Since this extension is purely based on commands it is not possible to add the screenshot of UI.

Exercise

To learn more about Commands API [1] and tabs.Execute [2], visit the MDN link given below.

We are having a small limitation on this. Our system will work when we have only one Youtube tab. But in the ideal world people can have more than one Youtube tabs. So instead of using the commands API, try to create the browserAction popup with the screenshot of the images. You can get the screenshot of the tab using **tabs.captureTab()** [3].

You can also do this exercise for other music streaming sites.

Optional: It will be great if you can share your code or blog about this learning on Twitter. Make sure you use hashtag #WebExtLearn when you are tweeting about this activity.

References

- [1] https://mzl.la/2zuxQoJ
- [2] https://mzl.la/2Lnf1aT
- [3] https://mzl.la/2D4QoPJ

Experiment 7: Building Quick Stackoverflow searcher

Problem Statement and Solution

Problem Statement

You know very well the number of times you have been to StackOverflow for searching throughout this day. Won't it be great to have a quick searcher for the StackOverflow? In this experiment, we will be building a simple Extension where we can type the shortcut in the newtab and the text to be searched so we will be directly taken to the search page.

Solution

One of the solutions we can see which is similar to both chrome and Firefox is below.

Step 1: Visit https://stackoverflow.com/search?q= Step 2: Bookmark this page. Step 3: Edit the property of the bookmark. Step 4: Give the shortcut (Keyword) for this bookmark in the bookmark manager.

Another solution using WebExtension, which is more similar to the mentioned above. We can do a lot of things using this. We will be defining the keyword first, and in the newtab when we are giving the keyword space the search string and hit enter we will be going to StackOverflow search page. Using WebExtension we are just replicating the above-mentioned steps.

How we can capture this

The address bar in the browser is called as Omnibox. In WebExtension's we are having specific API to handle the Omnibox. With this we can capture the letters being typed after our specified keyword, we can provide a suggestion as we start typing.

Note: The very important thing we should note here is the keyword, when the same keyword is used by another WebExtension then the extension which is installed at last will be taking over the control.

Building Blocks of Our Quick Stackoverflow searcher

Define the Keyword

Our first step is to define the right keyword. Our **keyword** should be very unique and so it won't be used by other or taken by other WebExtensions. Our keyword is defined in the manifest.json

```
  "omnibox": {
    "keyword": "sof"
  }
```

Adding Listener to our keyword

Once we type our keyword of the extension and space the characters what we are typing will be searching string. We can capture this (but it's not mandatory) and can show the default search suggestion.

```
  browser.omnibox.onInputChanged.addListener(text_typed);

  function text_typed(typed_text, suggest){
   // Do something
  }
```

So the listener callback will be giving two arguments, first is the text which is being typed and second is the suggest callback method.

Adding Listener when text is entered

This listener is mandatory for this extension. When the user types the string and then hits enter we need to take the control and do the action we have defined. So **omnibox.onInputEntered** is mandatory for this type of extension. All the post action is done here. The listener call back method will have two arguments first is the entered string and second is disposition object.

```
browser.omnibox.onInputEntered.addListener(text_entered);

function text_entered(entered_text, disposition) {
    // Do something
}
```

Assembling our Power Search Extension Parts

manifest.json file

In our manifest.json file we are making almost everything as usual, the additional thing we have added is for giving the keyword for this extension.

```
{
  "name": "Search in Stackoverflow",
  "description": "Shortcut to search quickly in StackOverFlow",
  "version": "1.0",
  "applications": {
      "gecko": {
          "strict_min_version": "52.0a1"
      }
  },
  "omnibox": {
      "keyword": "sof"
  },
  "background": {
      "scripts": ["background.js"]
  },
  "permissions": [
      "tabs"
  ],
  "manifest_version": 2
}
```

We have already discussed about the **omnibox** JSON, here is where we will be defining the keyword for this extension, and this is the most important part. The only part which will look different to you should be the below code, which sets the value of **strict_min_version**

```
"applications": {
      "gecko": {
          "strict_min_version": "52.0a1"
      }
  }
```

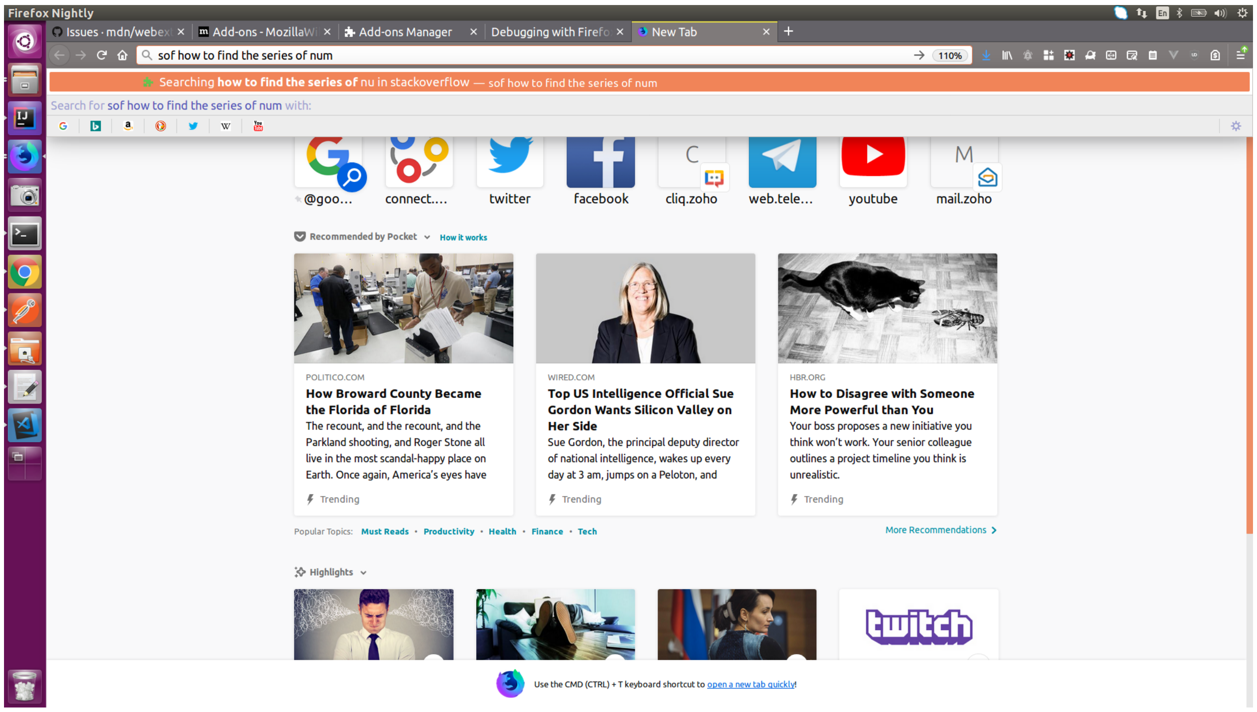
The meaning of above code is this extension will work only if the browser is having the version greater than *52.0a1*, this is very important. Some of the API's are introduced very late (after 57) or will be upcoming API's. We should make sure the user who is going to install our extension is having minimum the browser version from which our API is supported; so we can avoid the breakage in extension usage. Similarly, we are also having **strict_max_version** which will say the maximum version this extension will work. Generally, there was no much need for setting *strict_max_version* but we have to be careful when the major version of the browser is released.

Our Background Script File

Our Simple **background.js** file is shared below

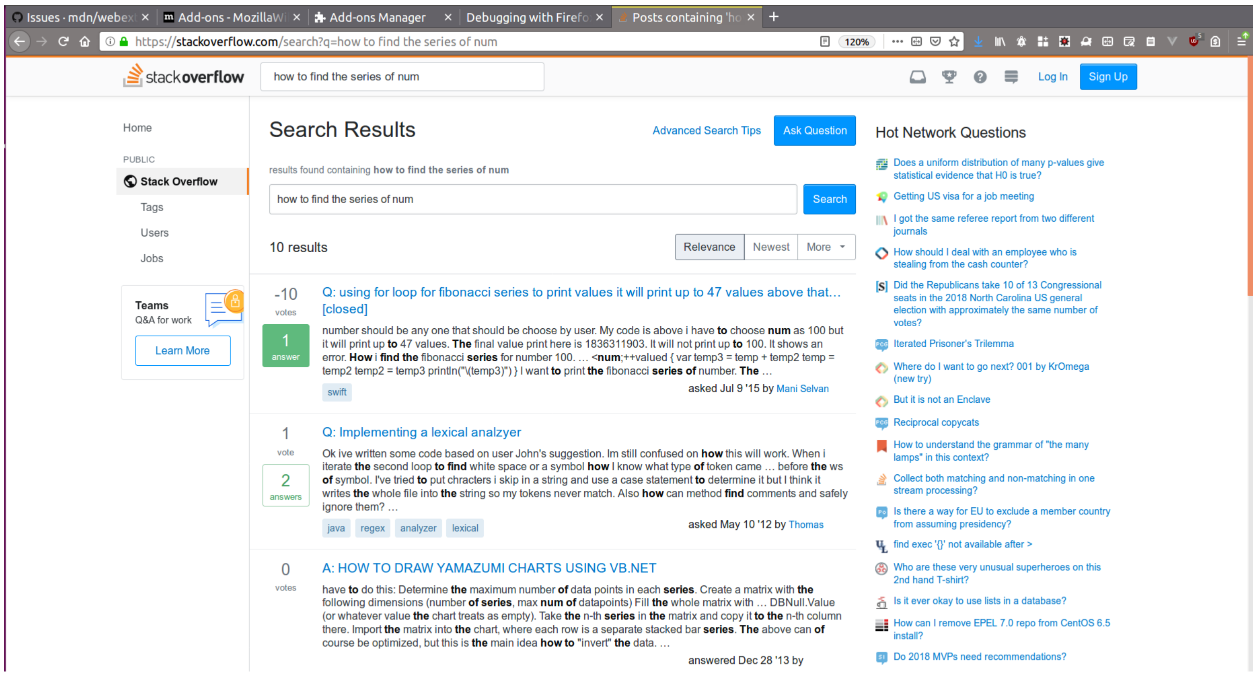
```
browser.omnibox.onInputChanged.addListener(text_typed);

browser.omnibox.onInputEntered.addListener(text_entered);

function text_entered(entered_text, disposition) {
    var url = 'https://stackoverflow.com/search?q=';
    url += entered_text;

    browser.tabs.update({
        url: url
    });

}

function text_typed(typed_text, suggest){
  browser.omnibox.setDefaultSuggestion({
    description: 'Searching '+typed_text+' in stackoverflow'
  });
}
```

So our first task is to listen whether our keyword is typed or not. If our keyword is typed then Firefox will activate our background script file (**background.js**). Then we will be listening to the input given by the users. Once the user starts giving the input we will be listening at **onInputChanged.addListener** and we will be passing the control to **text_typed** method, there we will be showing the list of suggestions.


Searching how to find the series of nu in stackoverflow — sof how to find the series of num
Search for sof how to find the series of num with:

Input Being Typed

Up next, the user will be finishing his/her typing and hits the enter. We will be listening to the end of the input given using the listener **onInputEntered.addListener** and we will be passing the control to **text_entered** method. In this method, we will be getting the **entered_text** variable. We simply have to append the URL and update the tab with new URL.



https://stackoverflow.com/search?q=how to find the series of num

how to find the series of num

Search Results

results found containing **how to find the series of num**

how to find the series of num

10 results

-10 votes, 1 answer

Q: using for loop for fibonacci series to print values it will print up to 47 values above that… [closed]

number should be any one that should be choose by user. My code is above i have **to** choose **num** as 100 but it will print up **to** 47 values. **The** final value print here is 1836311903. It will not print up **to** 100. It shows an error. **How** i **find the** fibonacci **series** for number 100. … <**num**;++valued { var temp3 = temp + temp2 temp = temp2 temp2 = temp3 println("\(temp3)") } I want **to** print **the** fibonacci **series of** number. **The** …

swift

asked Jul 9 '15 by Mani Selvan

1 vote, 2 answers

Q: Implementing a lexical analzyer

Ok ive written some code based on user John's suggestion. Im still confused on **how** this will work. When i iterate **the** second loop **to find** white space or a symbol **how** I know what type **of** token came … before **the** ws **of** symbol. I've tried **to** put chracters i skip in a string and use a case statement **to** determine it but I think it writes **the** whole file into **the** string so my tokens never match. Also **how** can method **find** comments and safely ignore them? …

java regex analyzer lexical

asked May 10 '12 by Thomas

0 votes

A: HOW TO DRAW YAMAZUMI CHARTS USING VB.NET

have **to** do this: Determine **the** maximum number **of** data points in each **series**. Create a matrix with **the** following dimensions (number **of series**, max **num of** datapoints) Fill **the** whole matrix with … DBNull.Value (or whatever value **the** chart treats as empty). Take **the** n-th **series** in **the** matrix and copy it **to the** n-th column there. Import **the** matrix into **the** chart, where each row is a separate stacked bar **series**. **The** above can **of** course be optimized, but this is **the** main idea **how to** "invert" **the** data. …

answered Dec 28 '13 by

Hot Network Questions

- Does a uniform distribution of many p-values give statistical evidence that H0 is true?
- Getting US visa for a job meeting
- I got the same referee report from two different journals
- How should I deal with an employee who is stealing from the cash counter?
- Did the Republicans take 10 of 13 Congressional seats in the 2018 North Carolina US general election with approximately the same number of votes?
- Iterated Prisoner's Trilemma
- Where do I want to go next? 001 by KrOmega (new try)
- But it is not an Enclave
- Reciprocal copycats
- How to understand the grammar of "the many lamps" in this context?
- Collect both matching and non-matching in one stream processing?
- Is there a way for EU to exclude a member country from assuming presidency?
- find exec '{}' not available after >
- Who are these very unusual superheroes on this 2nd hand T-shirt?
- Is it ever okay to use lists in a database?
- How can I remove EPEL 7.0 repo from CentOS 6.5 install?
- Do 2018 MVPs need recommendations?

Response at Stackoverflow

From the response page we can checkout the various solutions and find the best one for us.

Exercise

To learn More about below list of API, visit MDN link given below.

- `omnibox API` [1]
- `Tab Update API` [2]
- `SuggestResult API` [3]

Try to build your custom search engine, think of giving the generic keyword for activating Omnibox and then some text to search in some search engine, then finally search text. So with some combination of shortcuts, we can search in any search engine quickly. This exercise is slightly similar to the exercise what we had in our second experiment **Building our Power Search Add-on**, so its very easy and quick one.

Optional: It will be great if you can share your code or blog about this learning on Twitter. Make sure you use hashtag #WebExtLearn when you are tweeting about this activity.

References

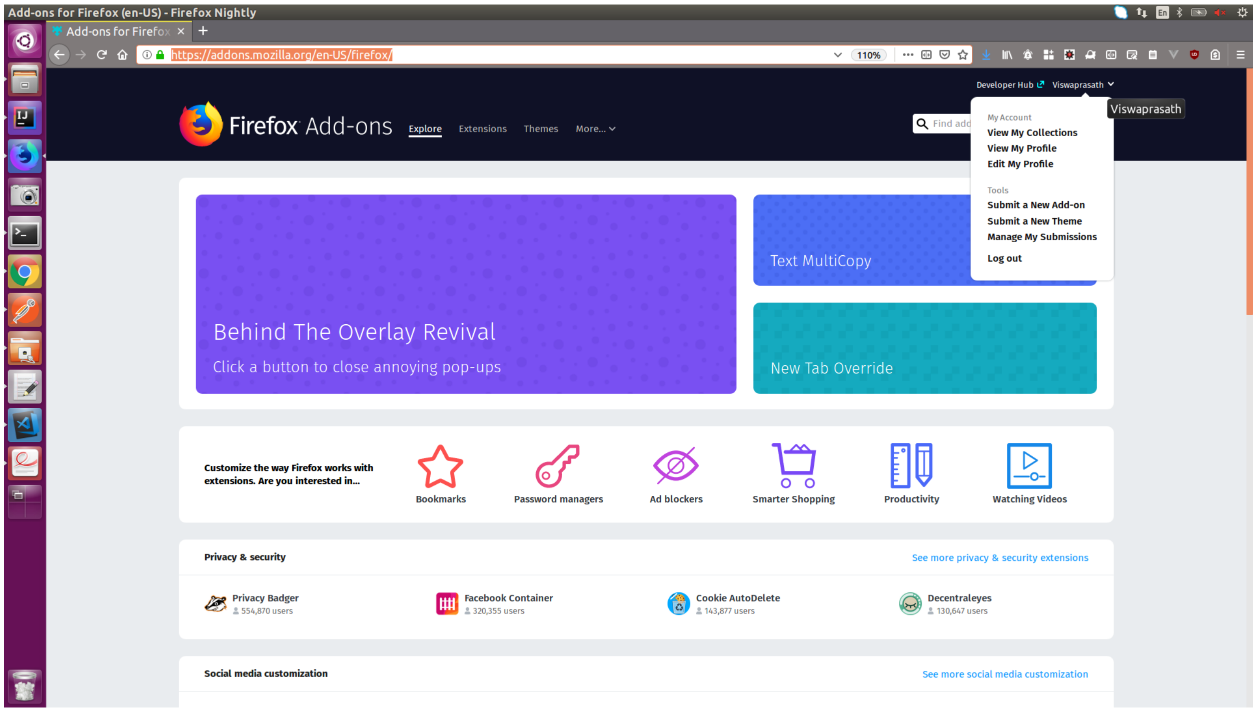
- [1] https://mzl.la/2PSO2dF

- [2] https://mzl.la/2Q2qYZM
- [3] https://mzl.la/2K4HpQ3

Publishing our First WebExtension

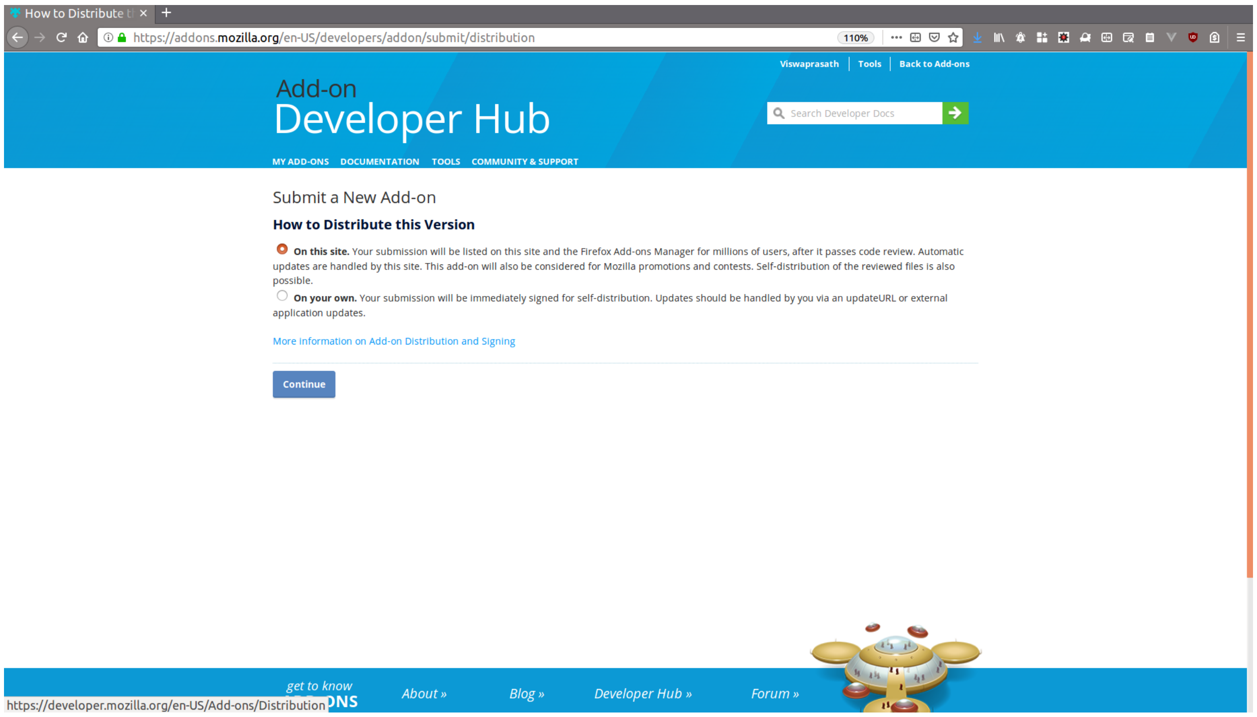
So we have started learning WebExtensions and our really learning excitement and lot of fun comes only when we start publishing our work. Publishing the developed Webextension in Firefox Add-ons store is very easy and its also free, there are some other browser specific stores which charges us. No need to worry about it.

- Step 1: Visit https://addons.mozilla.org/en-US/firefox/ (AMO website)
- Step 2: Click on Submit a New Add-on from the right top corner.



Clik on Submit a New Add-on

- Step 3: You will be landing to https://addons.mozilla.org/en-US/developers/addon/submit/distribution webpage.



Submit a New Add-on

How to Distribute this Version

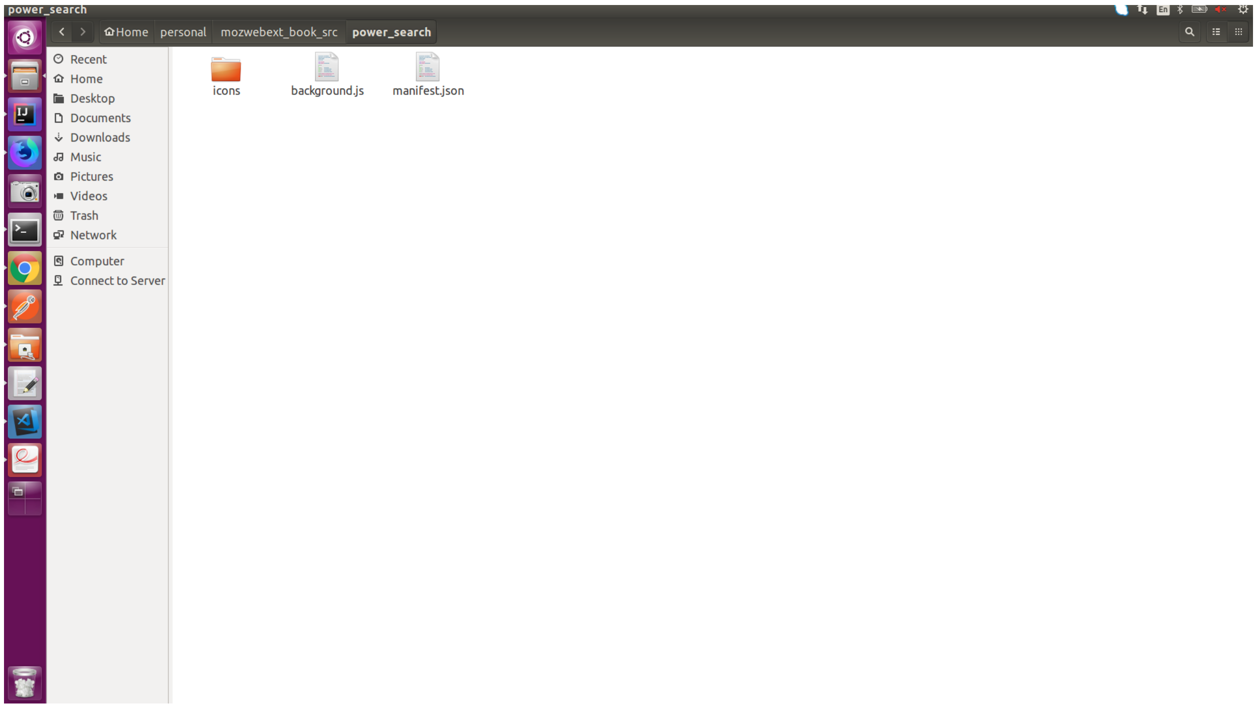
[✓ **On this site.**] Your submission will be listed on this site and the Firefox Add-ons Manager for millions of users, after it passes code review. Automatic updates are handled by this site. This add-on will also be considered for Mozilla promotions and contests. Self-distribution of the reviewed files is also possible.

On your own. Your submission will be immediately signed for self-distribution. Updates should be handled by you via an updateURL or external application updates.

More information on Add-on Distribution and Signing

Publish page

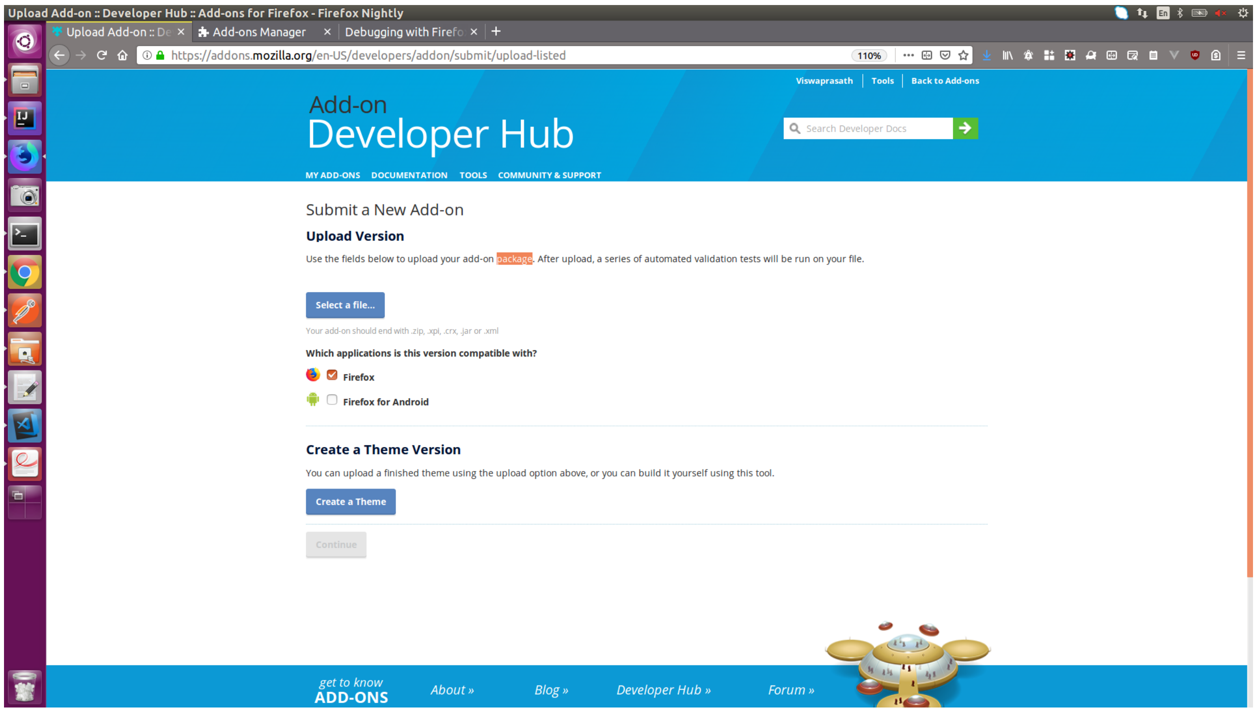
In this publish page we can now select **On this site** option. And click the next button. So before doing anything in AMO website, we should prepare our zip files of our extension.



Power Searcher Extension Contents

- We have to goto the directory where all our contents are present.
- Select every content in that directory.
- Compress as zip.
- So in our case **power_search.zip** is ready

Now we should be in uploader page.



Upload Add-on :: Developer Hub :: Add-ons for Firefox - Firefox Nightly

https://addons.mozilla.org/en-US/developers/addon/submit/upload-listed

Add-on Developer Hub

Submit a New Add-on

Upload Version

Use the fields below to upload your add-on package. After upload, a series of automated validation tests will be run on your file.

Your add-on should end with .zip, .xpi, .crx, .jar or .xml

Which applications is this version compatible with?

[✓] Firefox

[ ] Firefox for Android

Create a Theme Version

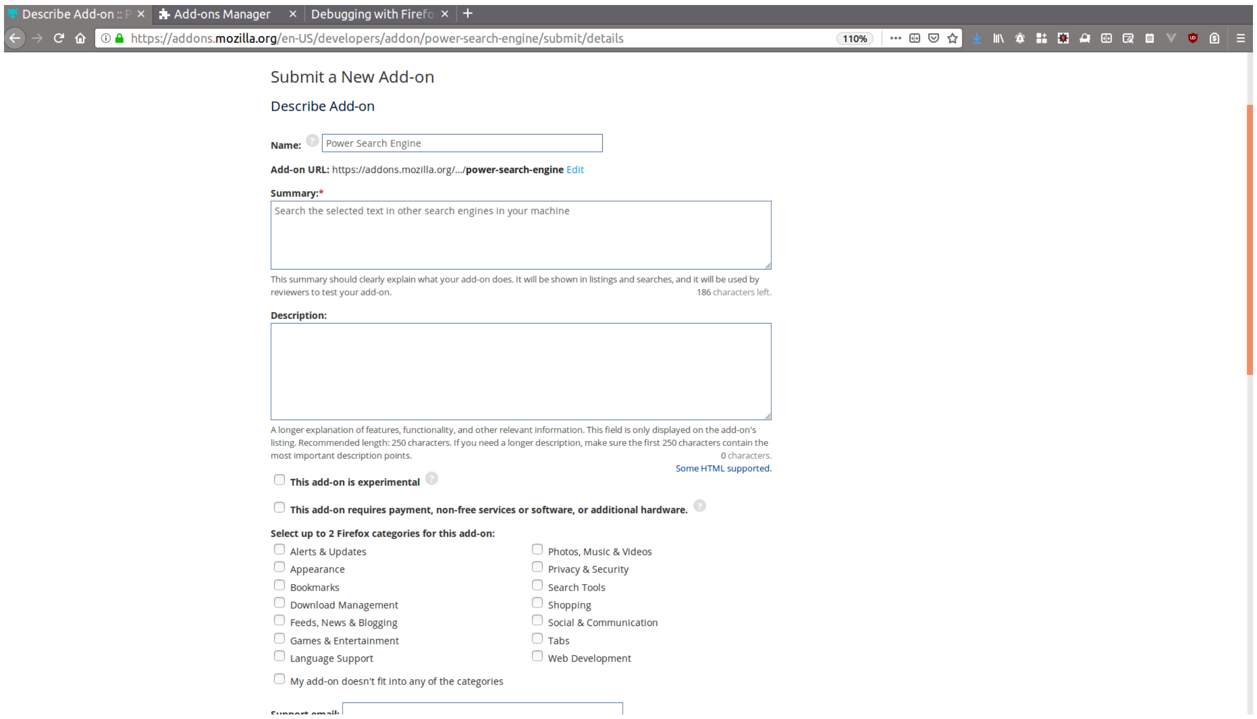
You can upload a finished theme using the upload option above, or you can build it yourself using this tool.

Zip uploader page

In this page we have to select the zip file which we have created just now. Once the upload is done, you will be seeing the continue button, click that.

The next page will ask **Do You Need to Submit Source Code?**, this is asked for developers who try using the JS minifier, template engine creator. As we are just getting started mostly our answer will be **no** for this.

The next page is you will be shown with the pre-populated values from your manifest.json



https://addons.mozilla.org/en-US/developers/addon/power-search-engine/submit/details

Submit a New Add-on

Describe Add-on

Name: Power Search Engine

Add-on URL: https://addons.mozilla.org/.../**power-search-engine** Edit

Summary:*

Search the selected text in other search engines in your machine

This summary should clearly explain what your add-on does. It will be shown in listings and searches, and it will be used by reviewers to test your add-on. 186 characters left.

Description:

A longer explanation of features, functionality, and other relevant information. This field is only displayed on the add-on's listing. Recommended length: 250 characters. If you need a longer description, make sure the first 250 characters contain the most important description points. 0 characters.

Some HTML supported.

☐ **This add-on is experimental**

☐ **This add-on requires payment, non-free services or software, or additional hardware.**

Select up to 2 Firefox categories for this add-on:

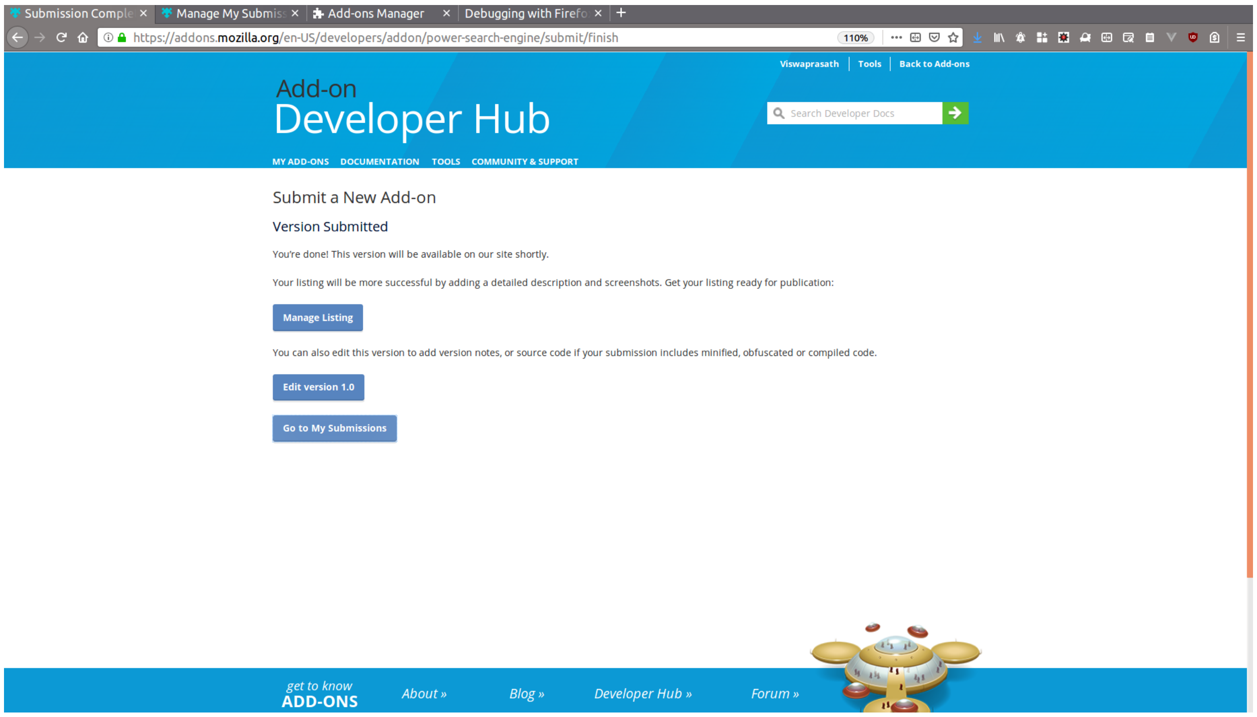
☐ Alerts & Updates	☐ Photos, Music & Videos
☐ Appearance	☐ Privacy & Security
☐ Bookmarks	☐ Search Tools
☐ Download Management	☐ Shopping
☐ Feeds, News & Blogging	☐ Social & Communication
☐ Games & Entertainment	☐ Tabs
☐ Language Support	☐ Web Development

☐ My add-on doesn't fit into any of the categories

Pre-populated values page

We can give the detailed description in this page. We have to select the categories, support email address and the license type of our add-ons. Then click on submit version.

You will be redirected to **version submitted** page



Submit a New Add-on

Version Submitted

You're done! This version will be available on our site shortly.

Your listing will be more successful by adding a detailed description and screenshots. Get your listing ready for publication:

You can also edit this version to add version notes, or source code if your submission includes minified, obfuscated or compiled code.

Version Submitted

If you click on **Manage Listing** button, you will be redirected to the page where you can edit the values for your extension.

Congrats

You have published your first WebExtension to AMO. With this, you can proudly announce you are Web extension developer. It's time for you to start sharing your extension development knowledge to others and make the community bigger and stronger.

Exercise

Publish one of the extension which you made during this journey. And share your code or blog about this learning on twitter or any social. Make sure you use hashtag #WebExtLearn when you are tweeting about this activity.