

افزونه چیست؟

افزونه‌ها جزء مهمترین قسمت‌های یک مرورگر توسعه پذیر به شمار می‌آیند. افزونه‌ها سعی دارند تا قابلیت‌هایی را به مرورگر شما اضافه کنند. افزونه‌ها از آخرین فناوری‌های HTML, CSS و جاوااسکریپت تا به آنجایی که مرورگر آن‌ها را پشتیبانی کند، استفاده می‌کنند.

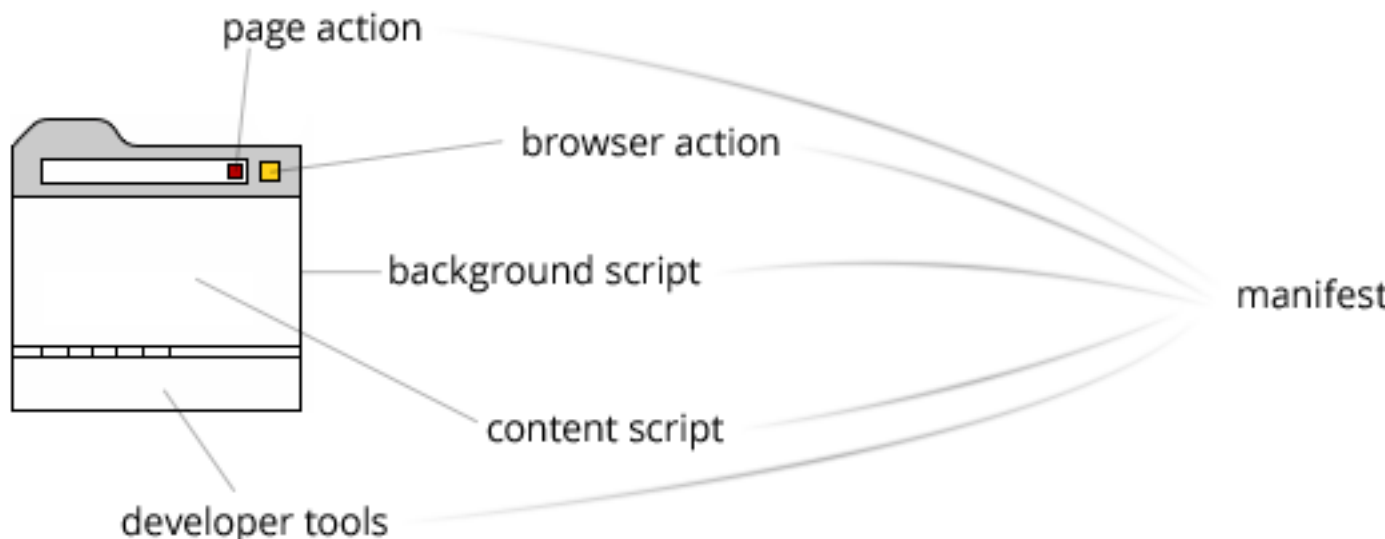
در این سری سعی خواهیم کرد برای هر مرورگر شناخته شده، یک افزونه ایجاد کنیم و ابتدا از آنجا که خودم از کروم استفاده می‌کنم، اولین افزونه را برای کروم خواهیم نوشت.

این افزونه قرار است چه کاری انجام دهد؟

کاری که برای این افزونه تدارک دیده‌ام این است: موقعی که سایت dotnettips.info به روز شد مرا آگاه کند. این آگاه‌سازی را از طریق یک نوتیفیکیشن به اطلاع کاربر می‌رسانیم. صفحه تنظیمات این افزونه شامل گزینه‌های "آخرین مطالب"، "نظرات آخرین مطالب"، "آخرین اشتراک‌ها" و "آخرین نظرات اشتراک‌ها" خواهد بود که به طور پیش فرض تنها گزینه اول فعال خواهد بود و همچنین یک گزینه نیز برای وارد کردن یک عدد صحیح جهت اینکه به افزونه بگوییم هر چند دقیقه یکبار سایت را چک کن. چک کردن سایت هم از طریق فید RSS صورت می‌گیرد.

فایل manifest.json

این فایل برای ذخیره سازی اطلاعاتی در مورد افزونه به کار می‌رود که شامل نام افزونه، توضیح کوتاه در مورد افزونه و ورژن و ... به کار می‌رود که همه این اطلاعات در قالب یا فرمت json نوشته می‌شوند و در بالاترین حد استفاده برای تعریف اهداف افزونه و اعطای مجوز به افزونه از آن استفاده می‌کنیم. این فایل بخش‌های زیر را در یک افزونه تعریف می‌کند که به مرور با آن آشنا می‌شویم.



کد زیر را در فایل manifest.json می‌نویسیم:

```
{
  "manifest_version": 2,
```

```

"name": "Dotnettips Updater",
"description": "This extension keeps you updated on current activities on dotnettips.info",
"version": "1.0",
"icons": { "16": "icon.png",
            "48": "icon.png",
            "128": "icon.png" },

"browser_action": {
  "default_icon": "icon.png",
  "default_popup": "popup.html"
},
"permissions": [
  "activeTab",
  "http://www.dotnettips.info"
]
}

```

اطلاعات اولیه شامل نام و توضیح و ورژن افزونه است. ورژن برنامه برای به روزآوری افزونه بسیار مهم است. موقعی که ورژن جدیدی از افزونه ارائه شود، گوگل وب استور اعلان آپدیت جدیدی را برای افزونه میکند. آیکن قسمت‌های مختلف افزونه هم با icons مشخص می‌شود که در سه اندازه باید ارائه شوند و البته اگر اندازه آن نباشد scale می‌شود. قسمت بعدی تعریف UI برنامه هست که گوگل کروم، به آن Browser Action می‌گوید. در اینجا یک آیکن و همچنین یک صفحه اختصاصی برای تنظیمات افزونه معرفی می‌کنیم. این آیکن کنار نوار آدرس نمایش داده می‌شود و صفحه popup موقعی نشان داده می‌شود که کاربر روی آن کلیک می‌کند. آیکن‌ها برای browser action در دو اندازه 19 و 38 پیکسلی هستند و در صورتی که تنها یک آیکن تعریف شود، به صورت خودکار عمل scale و تغییر اندازه صورت می‌گیرد. برای تعیین عکس برای هر اندازه می‌توانید کد را به صورت زیر بنویسید:

```

"default_icon": {
  // optional
  "19": "images/icon19.png", // optional
  "38": "images/icon38.png" // optional
}

```

قسمت popup برای نمایش تنظیمات به کار می‌رود و درست کردن این صفحه همانند صفحه همیشگی html هست و خروجی آن روی پنجره popup افزونه رندر خواهد شد.

گزینه default_title نیز یکی از دیگر خصیصه‌های مهم و پرکاربرد این قسمت هست که متن tooltip می‌باشد و موقعی که که کاربر، اشاره‌گر را روی آیکن ببرد نمایش داده می‌شود و در صورتی که نوشته نشود، کروم نام افزونه را نمایش می‌دهد؛ برای همین ما هم چیزی ننوشتیم.

صفحات پس‌زمینه

اگر بخواهید برای صفحه popup کد جاوااسکریپت بنویسید یا از jquery استفاده کنید، مانند هر صفحه‌ی وبی که درست می‌کنید آن را کنار فایل popup قرار داده و در popup آنها را صدا کرده و از آن‌ها استفاده کنید. ولی برای پردازش‌هایی که نیاز به UI وجود ندارد، می‌توان از صفحات پس‌زمینه استفاده کرد. در این حالت ما دو نوع صفحه داریم:

صفحات مصر یا Persistent Page

صفحات رویدادگرا یا Events Pages

اولین نوع صفحه، همواره فعال و در حال اجراست و دومی موقعی فعال می‌شود که به استفاده از آن نیاز است. گوگل توصیه می‌کند که تا جای ممکن از نوع دوم استفاده شود تا مقدار حافظه مصرفی حفظ شود و کارایی مرورگر بهبود بخشیده شود. کد زیر یک صفحه پس‌زمینه را از نوع رویدادگرا می‌سازد. به وضوح روشن است در صورتی که خاصیت Persistent با true مقداردهی شود، این صفحه مصرانه در تمام وقت باز بودن مرورگر، فعال خواهد بود:

```

"background": {
  "scripts": ["background.js"],
  "persistent": false
}

```

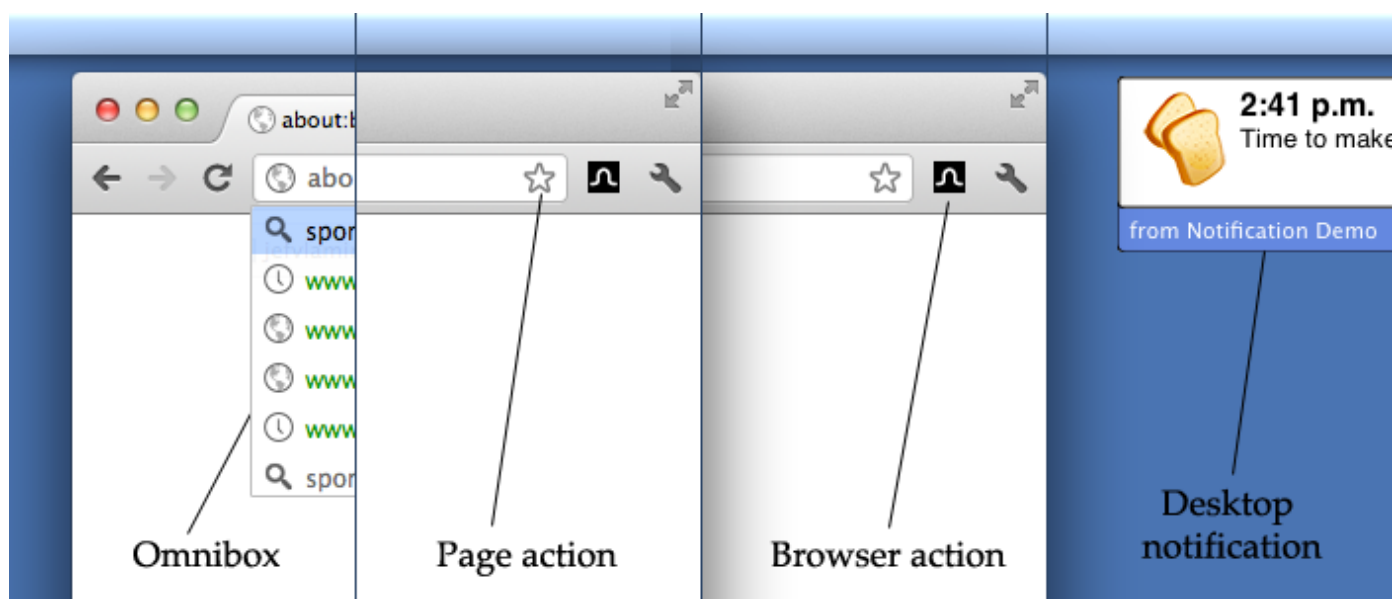
Content Script یا اسکریپت محتوا

در صورتی که بخواهید با هر صفحه‌ای که باز یا رفرش می‌شود، به DOM آن دسترسی پیدا کنید، از این خصوصیت استفاده کنید. در

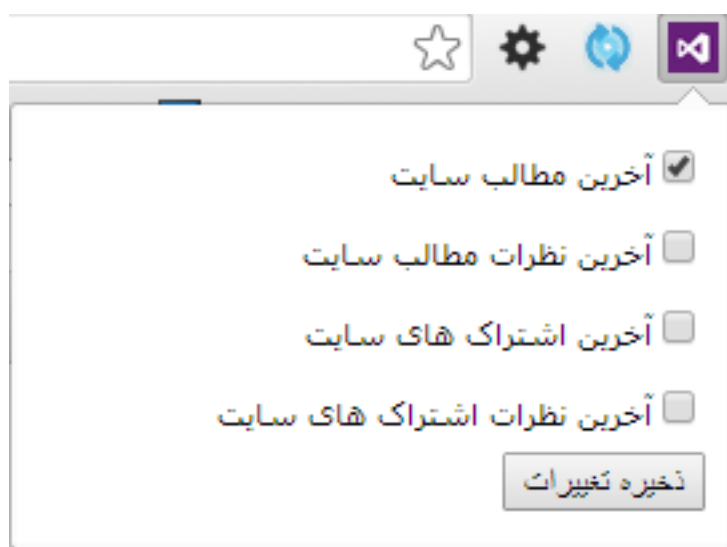
کد زیر برای پردازش اطلاعات DOM از فایل جاوااسکریپت بهره برده و در قسمت matches می‌گویید که چه صفحاتی باید از این کد استفاده کنند که در اینجا از پروتکل‌های HTTP استفاده میشود و اگر مثلاً نوع FTP یا file صدا زده شود کد مورد نظر اجرا نخواهد شد. در مورد اینکه matches چگونه کار می‌کند و چگونه می‌توان آن را نوشت، از این [صفحه](#) استفاده کنید.

```
"content_scripts": [  
  {  
    "matches": ["http://*/", "https://*/"],  
    "js": ["content.js"]  
  }  
]
```

آغاز کدنویسی (رابطهای کاربری)



اجازه دهید بقیه موارد را در حین کدنویسی تجربه کنیم و هر آنچه ماند را بعداً توضیح خواهیم داد. در اینجا من از یک صفحه با کد HTML زیر بهره برده ام که یک فرم دارد به همراه چهار چک باکس و در نهایت یک دکمه جهت ذخیره مقادیر. نام صفحه را popup.htm گذاشته ام و یک فایل popup.js هم دارم که در آن کد jquery نوشتم. قصد من این است که بتوان یک action browser به شکل زیر درست کنم:



کد html آن به شرح زیر است:

```
<html>
<head>
<meta charset="utf-8"/>

<script src="jquery.min.js"></script> <!-- Including jQuery -->
<script type="text/javascript" src="popup.js"></script>
</head>
<body style="direction:rtl;width:250px;">
<form >
<input type="checkbox" id="chkarticles" value="" checked="true">آخرین مطالب سایت</input><br/>
<input type="checkbox" id="chkarticlescomments" value="" >آخرین نظرات مطالب سایت</input><br/>
<input type="checkbox" id="chkshares" value="" >آخرین اشتراک‌های سایت</input><br/>
<input type="checkbox" id="chksharescomments" value="" >آخرین نظرات اشتراک‌های سایت</input><br/>
<input id="btnsave" type="button" value="ذخیره تغییرات" />
<div id="messageboard" style="color:green;"></div>
</form>

</body>
</html>
```

کد popup.js هم به شرح زیر است:

```
$(document).ready(function () {
    $("#btnsave").click(function() {
        var articles = $("#chkarticles").is(':checked');
        var articlesComments = $("#chkarticlescomments").is(':checked');
        var shares = $("#chkshares").is(':checked');
        var sharesComments = $("#chksharescomments").is(':checked');

        chrome.storage.local.set({ 'articles': articles, 'articlesComments': articlesComments,
        'shares': shares, 'sharesComments': sharesComments }, function() {
            $("#messageboard").text( 'تنظیمات جدید اعمال شد' );
        });
    });
});
```

در کد بالا موقعی که کاربر بر روی دکمه ذخیره، کلیک کند رویداد کلیک jquery فعال شده و مقادیر چک باکس‌ها را در متغیرهای مربوطه نگهداری می‌کند. نهایتاً با استفاده از کلمه کلیدی کروم به ناحیه ذخیره سازی داده‌های کروم دست پیدا کرده و درخواست ذخیره مقادیر چک باکس را بر اساس ساختار نام و مقدار، ذخیره می‌کنیم و بعد از اعمال، توسط یک تابع callback به کاربر اعلام می‌کنیم که اطلاعات ذخیره شده است.

اولین مورد جدیدی که در بالا دیدیم، کلمه‌ی کلیدی chrome است. کروم برای توسعه دهندگانی که قصد نوشتن افزونه دارند api هایی را تدارک دیده است که می‌توانید با استفاده از آنها به قسمت‌های مختلف مرورگر مثل بوک مارک یا تاریخچه فعالیت‌های مرورگر و ... دست پیدا کنید. البته برای اینکار باید در فایل manifest.json هم مجوز اینکار را درخواست نماییم. این ویژگی باید برای برنامه نویسان اندروید آشنا باشد. برای آشنایی هر چه بیشتر با مجوزها این [صفحه](#) را ببینید. برای دریافت مجوز، کد زیر را به manifest اضافه می‌کنیم:

```
"permissions": [
  "storage"
]
```

مجوزی که در بالا درخواست کرده‌ایم مجوز دسترسی به ناحیه ذخیره سازی است. بعد از کلمه کلیدی chrome، کلمه‌ی local آمده است و می‌گوید که باید داده‌ها به صورت محلی و لوکال ذخیره شوند ولی اگر میخواهید داده‌ها در گوگل سینک شوند، باید به جای لوکال از کلمه کلیدی sync استفاده کنید یعنی:

```
chrome.storage.sync.set
```

فایل manifest نهایی:

```
{
  "manifest_version": 2,
  "name": "Dotnettips Updater",
  "description": "This extension keeps you updated on current activities on dotnettips.info",
  "version": "1.0",

  "browser_action": {
    "default_icon": "icon.png",
    "default_popup": "popup.html"
  },
  "permissions": [
    "storage"
  ]
}
```

الان باید 4 فایل داشته باشید: فایل آیکن، popup.htm,popup.js و manifest.json. همه را داخل یک دایرکتوری قرار داده و در مرورگر کروم به قسمت extensions بروید و گزینه Developer mode را فعال کنید تا یک تستی از کد نوشته شده بگیرید. گزینه Load Unpacked Extension را بزنید و آدرس دایرکتوری ایجاد شده را به آن بدهید.

chrome://extensions

Extensions

Load unpacked extension...

Pack extension...

☒ Developer mode

Update extensions n...

الان باید مانند تصویر بالا یک آیکن کنار نوار آدرس یا به قول گوگل، Omni box ببینید. گزینه‌ها را تیک بزنید و روی دکمه ذخیره کلیک کنید. باید پیام مقادیر ذخیره شدند، نمایش پیدا کند. الان یک مشکل وجود دارد؛ داده‌ها ذخیره می‌شوند ولی موقعی که دوباره تنظیمات افزونه را باز کنید حالت اولیه نمایش داده می‌شود. پس باید تنظیمات ذخیره شده را خوانده و به آن‌ها اعمال کنیم. کد زیر را جهت دریافت مقادیر ذخیره شده می‌نویسیم. اینبار به جای استفاده از متد set از متد get استفاده می‌کنیم. به صورت آرایه، رشته نام مقادیر را درخواست می‌کنیم و در تابع callback، مقادیر به صورت آرایه برای ما برگشت داده می‌شوند.

```
chrome.storage.local.get(['articles', 'articlesComments', 'shares', 'sharesComments'], function (
items) {
  console.log(items[0]);
  $("#chkarticles").attr("checked", items["articles"]);
  $("#chkarticlescomments").attr("checked", items["articlesComments"]);
  $("#chkshares").attr("checked", items["shares"]);
  $("#chksharescomments").attr("checked", items["sharesComments"]);
});
```

حالا برای اینکه افزونه‌ی شما متوجه تغییرات شود، به تب extensions رفته و در لیست افزونه‌ها به دنبال افزونه خود بگردید و گزینه Reload را انتخاب نمایید تا افزونه تغییرات را متوجه شود و صفحه را تست کنید.

Page Action

روش دیگر برای ارائه یک رابط کاربری، page action هست. این روش دقیقا مانند روش قبلی است، ولی جای آیکن عوض می‌شود. قبلا بیرون از نوار آدرس بود، ولی الان داخل نوار آدرس قرار می‌گیرد. جالب‌ترین نکته در این مورد این است که این آیکن در ابتدا مخفی شده است و شما تصمیم می‌گیرید که این آیکن چه موقع نمایش داده شود. مثلا آیکن RSS تنها موقعی نمایش داده

می‌شود که وب سایتی که باز شده است، دارای محتوای RSS باشد یا بوک مارک کردن یک آدرس برای همه‌ی سایت‌ها باز باشد و سایر موارد.

کد زیر نحوه‌ی تعریف یک page action را در manifest نشان می‌دهد. ما در این مثال یک page action را به طور موقت اضافه می‌کنیم و موقعی هم آن را نشان می‌دهیم که سایت dotnettips.info باشد. دلیل اینکه موقت اضافه می‌کنیم این است که باید یکی از دو گزینه رابط کاربری که تا به حال گفتیم، استفاده شود. در غیر این صورت کروم در هنگام خواندن فایل manifest در هنگام افزودن افزونه به مرورگر، پیام خطا خواهد داد و این مطلب را به شما گوشزد می‌کند. پس نمی‌توان دو گزینه را همزمان داشت و من می‌خواهم افزونه را در حالت browser action ارائه کنم. پس در پروژه نهایی، این مطلب page action نخواهد بود. برای داشتن یک page action کد زیر را در manifest بنویسید.

```
"page_action": {
  "default_icon": {
    "19": "images/icon19.png",
    "38": "images/icon38.png"
  },
  "default_popup": "popup.html"
```

گزینه page action تعریف شد حالا باید کاری کنیم تا هر موقع صفحه‌ای باز می‌شود چک کند آیا سایت مورد نظر است یا خیر، اینکار را توسط صفحه‌ی پردازشی انجام می‌دهیم. پس تکه کد زیر را هم به manifest اضافه می‌کنیم:

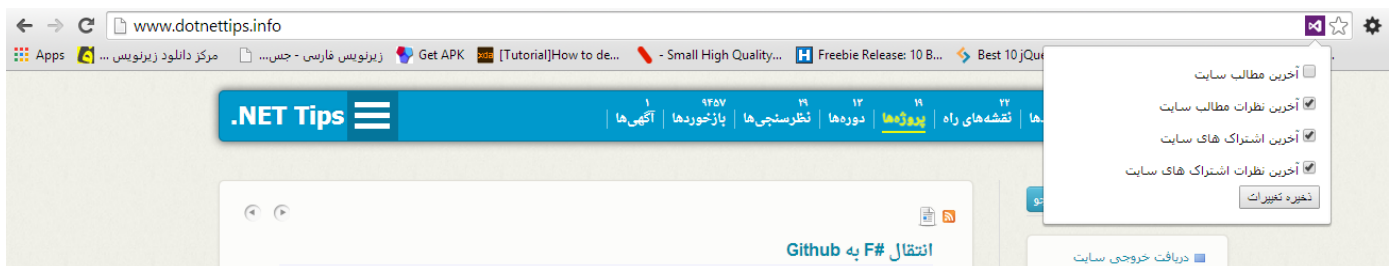
```
"background": {
  "scripts": ["page_action_validator.js"]
}
```

تا اینجا فایل جاوااسکریپت معرفی شد که کد زیر را دارد و در پس زمینه شروع به اجرا می‌کند.

```
function UrlValidation(tabId, changeInfo, tab) {
  if (tab.url.indexOf('dotnettips.info') > -1) {
    chrome.pageAction.show(tabId);
  }
};
chrome.tabs.onUpdated.addListener(UrlValidation);
```

چون از api در این کد بهره برده‌ایم و آن هم مدیریت بر روی تب هاست، پس باید مجوز آن هم گرفته شود. کلمه "tabs" را در قسمت permissions اضافه کنید.

یک listener برای tab‌ها ایجاد کرده‌ایم که اگر تب جدید ایجاد شد، یا تب قبلی به آدرس جدیدی تغییر پیدا کرد تابع UrlValidation را اجرا کند و در این تابع چک می‌کنیم که اگر url این تب شامل نام وب سایت می‌شود، page action روی این تب ظاهر شود. پس از انجام تغییرات، مجدداً افزونه را بارگذاری می‌کنیم و تغییرات اعمال شده را می‌بینیم. سایت dotnettips را باز کنید یا صفحه را مجدداً رفرش کنید تا تغییر اعمال شده را ببینید.



تغییرات موقت را حذف و کدها را به حالت قبلی یعنی browser action برگردانیم.

omnibox یک کلمه کلیدی است که در نوار آدرس مرورگر وارد می‌شود و در واقع می‌توانیم آن را نوع دیگری از رابط کاربری بنامیم. موقعی که شما کلمه کلیدی رزرو شده را وارد می‌کنید، در نوار آدرس کلماتی نشان داده می‌شود که کاربر می‌تواند یکی از آن‌ها را انتخاب کند تا عملی انجام شود. ما هم قرار است این کار را انجام دهیم. به این مثال دقت کنید:

می‌خواهیم موقعی که کاربر کلمه net را تایپ می‌کند، 5 عبارت آخرین مطالب و آخرین اشتراک‌ها و آخرین نظرات مطالب و آخرین نظرات اشتراک‌ها و صفحه اصلی سایت نمایش داده شود و با انتخاب هر کدام، کاربر به سمت آن صفحه هدایت شود.

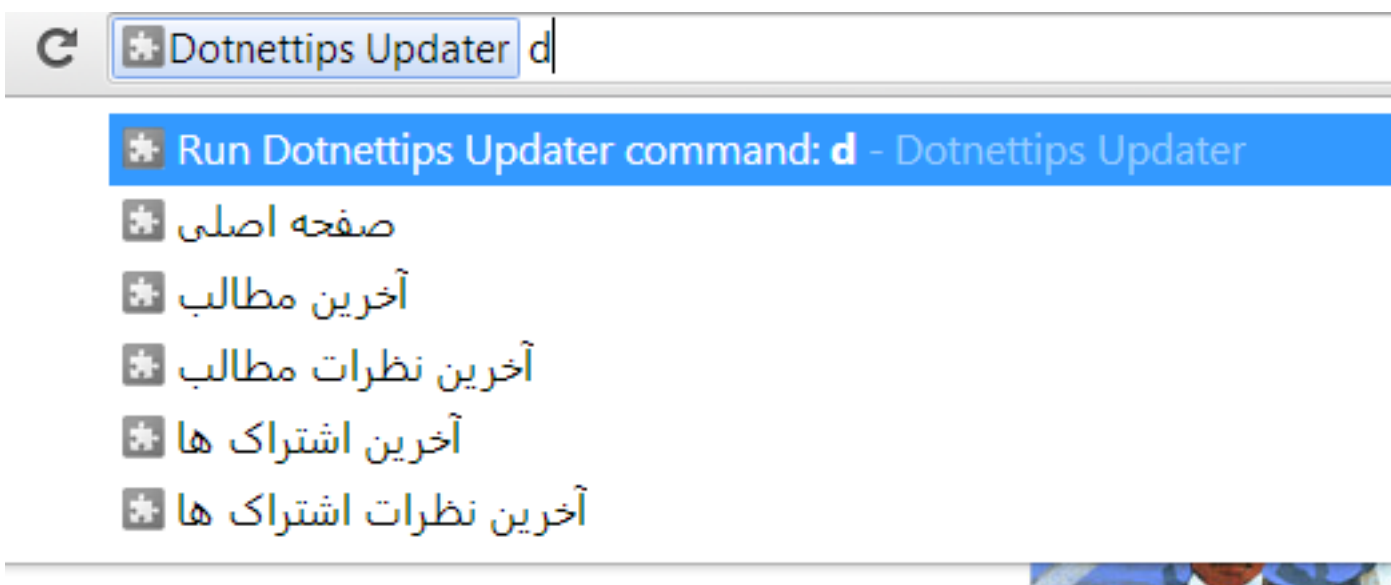
برای افزودن کلمه کلیدی در manifest خطوط زیر را اضافه کنید:

```
"omnibox": { "keyword" : ".net" }
```

با نوشتن خط بالا کلمه net در مرورگر یک کلمه‌ی کلیدی به حساب خواهد آمد و موقعی که کاربر این کلمه را وارد کند، در سمت راست نوشته خواهد شد. در این حالت باید کلید تب را بزند تا به محیط دستوری آن برود.

Press **Tab** to send commands to Dotnettips Updater

در این حین می‌توانیم همزمان با تایپ کاربر، دستوراتی را به آن نشان بدهیم. من دوست دارم موقعی که کاربر حرفی را وارد کرد، لیستی از نام صفحات نوشته شود.



برای اینکار باید کدنویسی کنیم ، پس یک فایل پس زمینه را به manifest معرفی کنید:

```
"background": {  
  "scripts": ["omnibox.js"]  
}
```

در فایل ominibox.js دستوراتی که مرتبط با omnibox است را می‌نویسیم و کد زیر را به آن اضافه می‌کنیم:

```
chrome.omnibox.onInputChanged.addListener(function(text, suggest) {  
  suggest([  
    {content: ".net tips Home Page", description: "صفحه اصلی"},  
    {content: ".net tips Posts", description: "آخرین مطالب"},  
    {content: ".net tips News", description: "آخرین نظرات مطالب"},  
    {content: ".net tips Post Comments", description: "آخرین اشتراک ها"},  
    {content: ".net tips News Comments", description: "آخرین نظرات اشتراک ها"}  
  ]);  
});
```

chrome.omnibox شامل 4 رویداد می‌شود:

بعد از اینکه کاربر کلمه کلیدی را وارد کرد اجرا می‌شود	onInputStarted
بعد از وارد کردن کلمه کلیدی هربار که کاربر تغییری در ورودی نوارد آدرس می‌دهد اجرا می‌شود.	onInputChanged
کاربر ورودی خود را تایید می‌کند. مثلا بعد از وارد کردن، کلید enter را می‌فشارد	onInputEntered
کاربر از وارد کردن ورودی منصرف شده است؛ مثلا کلید ESC را فشرده است.	onInputCancelled

با نوشتن `chrome.omnibox.onInputChanged.addListener` ما یک listener ساخته‌ایم تا هر بار کاربر ورودی را تغییر داد، یک تابع callback که دو آرگومان را دارد، صدا بزند. این آرگومان‌ها یکی متن ورودی است و دیگری آرایه‌ی `suggest` که شما با تغییر آرایه می‌توانید عباراتی که همزمان با تایپ به کاربر پیشنهاد می‌شود را نشان دهید. البته می‌توانید با تغییر کد کاری کنید تا بر اساس حروفی که تا به حال تایپ کرده‌اید، دستورات را نشان دهد؛ ولی من به دلیل اینکه 5 دستور بیشتر نبود و کاربر راحت باشد، چنین کاری نکردم. همچنین وقتی شما برای هر یک `description` تعریف کنید، به جای نام پیشنهادی، توضیح آن را نمایش می‌دهد. حالا وقت این است که کد زیر را جهت اینکه اگر کاربر یکی از کلمات پیشنهادی را انتخاب کرد، به صفحه‌ی مورد نظر هدایت شود، اضافه کنیم:

```
chrome.omnibox.onInputEntered.addListener(function (text) {
  var location="";
  switch(text)
  {
    case ".net tips Posts":
      location="http://www.dotnettips.info/postsarchive";
      break;
    case ".net tips News":
      location="http://www.dotnettips.info/newsarchive";
      break;
    case ".net tips Post Comments":
      location="http://www.dotnettips.info/commentsarchive";
      break;
    case ".net tips News Comments":
      location="http://www.dotnettips.info/newsarchive/comments";
      break;
    default:
      location="http://www.dotnettips.info/";
  }

  chrome.tabs.getSelected(null, function (tab) {
    chrome.tabs.update(tab.id, { url: location });
  });
});
```

ابتدا یک listener برای روی رویداد `onInputEntered` قرار داده تا وقتی کاربر عبارت وارد شده را تایید کرد، اجرا شود. در مرحله بعد چک می‌کنیم که عبارت وارد شده چیست و به ازای هر عبارت مشخص شده، آدرس آن صفحه را در متغیر `location` قرار می‌دهیم. در نهایت با استفاده از عبارت `chrome.tabs.getSelected` تب انتخابی را به یک تابع callback بر میگردانیم. اولین آرگومان `windowId` است، برای زمانی که چند پنجره کروم باز است که می‌توانید وارد نکنید تا پنجره فعلی و تب فعلی محسوب شود. برای همین نال رد کردیم. در تابع برگشتی، شیء `tab` شامل اطلاعات کاملی از آن تب مانند `url` و `id` و `title` می‌باشد و در نهایت با استفاده از دستور `chrome.tabs.update` اطلاعات تب را به روز می‌کنیم. آرگومان اول `id` تب را می‌دهیم تا بداند کدام تب باید تغییر کند و آرگومان بعدی می‌توانید هر یک از ویژگی‌های تب از قبیل آدرس فعلی یا عنوان آن و ... را تغییر دهید که ما آدرس آن را تغییر داده ایم.

یکی دیگر از رابطهای کاربری، منوی کانتکست هست که توسط chrome.contextmenus ارائه می‌شود و به مجوز "contextmenus" نیاز دارد. فعال سازی منوی کانتکست در قسمت‌های زیر ممکن است:

all, page, frame, selection, link, editable, image, video, audio

من گزینه‌ی dotenettips.info را برای باز کردن سایت، به Contextmenus اضافه می‌کنم. کد را در فایلی به اسم contextmenus.js ایجاد می‌کنم و در قسمت background آن را معرفی می‌کنم. برای باز کردن یک تب جدید برای سایت، نیاز به chrome.tabs داریم که البته نیاز به مجوز tabs هم داریم.
محتوای فایل contextmenus.js

```
var root = chrome.contextMenus.create({
  title: 'Open .net tips',
  contexts: ['page']
}, function () {
  var Home= chrome.contextMenus.create({
    title: 'Home',
    contexts: ['page'],
    parentId: root,
    onclick: function (evt) {
      chrome.tabs.create({ url: 'http://www.dotnettips.info' })
    }
  });
  var Posts = chrome.contextMenus.create({
    title: 'Posts',
    contexts: ['page'],
    parentId: root,
    onclick: function (evt) {
      chrome.tabs.create({ url: 'http://www.dotnettips.info/postsarchive/' })
    }
  });
});
```

در کد بالا یک گزینه به context menu اضافه میشود و دو زیر منو هم دارد که یکی صفحه‌ی اصلی سایت را باز میکند و دیگری هم صفحه‌ی مطالب سایت را باز می‌کند.

تا به اینجا ما قسمت ظاهری کار را آماده کرده ایم و به دلیل اینکه مطلب طولانی نشود، این مطلب را در دو قسمت ارائه خواهیم کرد. در قسمت بعدی نحوه خواندن RSS و اطلاع رسانی و دیگر موارد را بررسی خواهیم کرد.

[در مقاله پیشین](#) ما ظاهر افزونه را طراحی و یک سری از قابلیت‌های افزونه را معرفی کردیم. در این قسمت قصد داریم پردازش پس زمینه افزونه یعنی خواندن RSS و اعلام به روز آوری سایت را مورد بررسی قرار دهیم و یک سری قابلیت هایی که گوگل در اختیار ما قرار داده است.

خواندن RSS توسط API های گوگل

گوگل در تعدادی از زمینه‌ها و سرویس‌های خودش [api های](#) را ارائه کرده است که یکی از آن‌ها [خواندن فید](#) است و ما از آن برای خواندن RSS یا اتم وب سایت کمک می‌گیریم. روند کار بدین صورت است که ابتدا ما بررسی می‌کنیم کاربر چه مقادیری را ثبت کرده است و افزونه قرار است چه بخش هایی از وب سایت را بررسی نماید. در این حین، صفحه پس زمینه شروع به کار کرده و در هر سیکل زمانی مشخص شده بررسی می‌کند که آخرین بار چه زمانی RSS به روز شده است. اگر از تاریخ قبلی بزرگتر باشد، پس سایت به روز شده است و تاریخ جدید را برای دفعات آینده جایگزین تاریخ قبلی کرده و یک پیام را به صورت نوتیفیکیشن جهت اعلام به روز رسانی جدید در آن بخش به کاربر نشان می‌دهد.

اجازه دهید کدها را کمی شکل‌تر کنیم. من از فایل زیر که یک فایل جاوااسکریپتی است برای نگه داشتن مقادیر بهره می‌برم تا اگر روزی خواستم یکی از آن‌ها را تغییر دهم راحت باشم و در همه جا نیاز به تغییر مجدد نداشته باشم. نام فایل را (const.js) به خاطر ثابت بودن آن‌ها انتخاب کرده‌ام.

```
// برای ذخیره مقادیر از ساختار نام و مقدار استفاده می‌کنیم که نام‌ها را اینجا ثبت کرده ام
var Variables={
  posts:"posts",
  postsComments:"postsComments",
  shares:"shares",
  sharesComments:"sharesComments",
}

// برای ذخیره زمان آخرین تغییر سایت برای هر یک از مطالب به صورت جداگانه نیاز به یک ساختار نام و مقدار
// است که نام‌ها را در اینجا ذخیره کرده ام
var DateContainer={
  posts:"dtposts",
  postsComments:"dtpostsComments",
  shares:"dtshares",
  sharesComments:"dtsharesComments",
  interval:"interval"
}

// برای نمایش پیام‌ها به کاربر
var Messages={
  SettingsSaved:"تنظیمات ذخیره شد",
  SiteUpdated:"سایت به روز شد",
  PostsUpdated:"مطلب ارسالی جدید به سایت اضافه شد",
  CommentsUpdated:"نظری جدیدی در مورد مطالب سایت ارسال شد",
  SharesUpdated:"اشتراک جدید به سایت ارسال شد",
  SharesCommentsUpdated:"نظری برای اشتراک‌های سایت اضافه شد"
}

// لینک‌های فید سایت
var Links={
  postUrl:"http://www.dotnettips.info/feeds/posts",
  posts_commentsUrl:"http://www.dotnettips.info/feeds/comments",
  sharesUrl:"http://www.dotnettips.info/feed/news",
  shares_commentsUrl:"http://www.dotnettips.info/feed/newscomments"
}

// لینک صفحات سایت
var WebLinks={
  Home:"http://www.dotnettips.info",
  postUrl:"http://www.dotnettips.info/postsarchive",
  posts_commentsUrl:"http://www.dotnettips.info/commentsarchive",
  sharesUrl:"http://www.dotnettips.info/newsarchive",
  shares_commentsUrl:"http://www.dotnettips.info/newsarchive/comments"
}
```

موقعی که اولین بار افزونه نصب می‌شود، باید مقادیر پیش فرضی وجود داشته باشند که یکی از آن‌ها مربوط به مقدار سیکل زمانی

است (هر چند وقت یکبار فید را چک کند) و دیگری ذخیره مقادیر پیش فرض رابط کاربری که قسمت پیشین درست کردیم؛
پروسه پس زمینه برای کار خود به آن‌ها نیاز دارد و بعدی هم تاریخ نصب افزونه است برای اینکه تاریخ آخرین تغییر سایت را با آن مقایسه کند که البته با اولین به روزرسانی تاریخ فید جای آن را می‌گیرد. جهت انجام اینکار یک فایل `init.js` ایجاد کرده‌ام که قرار است بعد از نصب افزونه، مقادیر پیش فرض بالا را ذخیره کنیم.

```
chrome.runtime.onInstalled.addListener(function(details) {
var now=String(new Date());

var params={};
params[Variables.posts]=true;
params[Variables.postsComments]=false;
params[Variables.shares]=false;
params[Variables.sharesComments]=false;

params[DateContainer.interval]=1;

params[DateContainer.posts]=now;
params[DateContainer.postsComments]=now;
params[DateContainer.shares]=now;
params[DateContainer.sharesComments]=now;

chrome.storage.local.set(params, function() {
  if(chrome.runtime.lastError)
  {
    /* error */
    console.log(chrome.runtime.lastError.message);
    return;
  }
});
});
```

[chrome.runtime](#) شامل رویدادهایی چون `onInstalled` , `onStartup` , `onSuspend` و ... است که مربوطه به وضعیت اجرایی افزونه میشود. آنچه ما اضافه کردیم یک `listener` برای زمانی است که افزونه نصب شده است و در آن مقادیر پیش فرض ذخیره می‌شوند. اگر خوب دقت کنید می‌بینید که روش ذخیره سازی ما در اینجا کمی متفاوت از مقاله پیشین هست و شاید پیش خودتان بگویید که احتمالا به دلیل زیباتر شدن کد اینگونه نوشته شده است ولی مهمترین دلیل این نوع نوشتار این است که متغیرهای بین { } آنچنان فرقی با خود `string` نمی‌کنند یعنی کد زیر:

```
chrome.storage.local.set('mykey':myvalue,....
```

با کد زیر برابر است:

```
chrome.storage.local.set(mykey:myvalue,...
```

پس اگر مقداری را داخل متغیر بگذاریم آن مقدار حساب نمی‌شود؛ بلکه کلید نام متغیر خواهد شد.
برای معرفی این دو فایل `const.js` و `init.js` به `manifest.json` می‌توانید به صورت زیر عمل کنید:

```
"background": {
  "scripts": ["const.js","init.js"]
}
```

در این حالت خود اکستنشن در زمان نصب یک فایل `html` درست کرده و این دو فایل `js` را در آن صدا می‌زنند که البته خود ما هم می‌توانیم اینکار را مستقیما انجام دهیم. مزیت اینکه ما خودمان مسقیما این کار را انجام دهیم این است که در صورتی که فایل‌های `js` ما زیاد شوند، فایل `manifest.json` زیادی شلوغ شده و شکل زشتی پیدا می‌کند و بهتر است این فایل را تا آنجا که می‌توانیم خلاصه نگه داریم. البته روش بالا برای دو یا سه تا فایل `js` بسیار خوب است ولی اگر به فرض بشود 10 تا یا بیشتر بهتر است یک فایل جداگانه شود و من به همین علت فایل `background.htm` را درست کرده و به صورت زیر تعریف کرده‌ام:

نکته: نمی‌توان در تعریف بک گراند هم فایل اسکریپت معرفی کرد و هم فایل `html`

```
"background": {
  "page": "background.htm"
}
```

```
<html>
<head>
  <script type="text/javascript" src="const.js"></script>
  <script type="text/javascript" src="https://www.google.com/jsapi"></script>
  <script type="text/javascript" src="init.js"></script>
<script type="text/javascript" src="omnibox.js"></script>
<script type="text/javascript" src="rssreader.js"></script>
<script type="text/javascript" src="contextmenus.js"></script>
</head>
<body>
</body>
</html>
```

لینک‌های بالا به ترتیب معرفی ثابت‌ها، لینک api گوگل که بعداً بررسی می‌شود، فایل init.js برای ذخیره مقادیر پیش فرض، فایل ominibox که در مقاله پیشین در مورد آن صحبت کردیم و فایل rssreader.js که جهت خواندن rss در پایبتر در موردش بحث می‌کنیم و فایل contextmenus که این را هم در مطلب پیشین توضیح دادیم.

جهت خواندن فید سایت ما از Google API استفاده می‌کنیم؛ اینکار دو دلیل دارد:

کدنویسی راحت‌تر و خلاصه‌تر برای خواندن RSS

استفاده اجباری از یک پروکسی به خاطر [Content Security Policy](#) و حتی [CORS](#)

قبل از اینکه manifest به ورژن 2 برسد ما اجازه داشتیم کدهای جاوااسکریپت به صورت inline در فایل‌های html بنویسیم و یا اینکه از منابع و آدرس‌های خارجی استفاده کنیم برای مثال یک فایل jquery بر روی وب سایت [jquery](#)؛ ولی از ورژن 2 به بعد، گوگل سیاست امنیت محتوا Content Security Policy را که سورس و سند اصلی آن در [اینجا](#) قرار دارد، به سیستم Extension خود افزود تا از حملاتی قبیل XSS و یا تغییر منبع راه دور به عنوان یک malware جلوگیری کند. پس ما از این به بعد نه اجازه داشتیم inline بنویسیم و نه اجازه داشتیم فایل jquery را از روی سرورهای سایت سازنده صدا بزنیم. پس برای حل این مشکل، ابتدا مثل همیشه یک فایل js را در فایل html معرفی می‌کردیم و برای حل مشکل دوم باید منابع را به صورت محلی استفاده می‌کردیم؛ یعنی فایل jquery را داخل دایرکتوری extension قرار می‌دادیم.

برای حل مشکل صدا زدن فایل‌های راه دور ما از [Relaxing the Default Policy](#) استفاده می‌کنیم که به ما یک لیست سفید ارائه می‌کند و در این لیست سفید دو نکته‌ی مهم به چشم می‌خورد که یکی از آن این است که استفاده از آدرس‌هایی با پروتکل Https و آدرس لوکال local host/127.0.0.1 بلا مانع است و از آنجا که api گوگل یک آدرس Https است، می‌توانیم به راحتی از API آن استفاده کنیم. فقط نیاز است تا خط زیر را به manifest.json اضافه کنیم تا این استثناء را برای ما در نظر بگیرد.

```
"content_security_policy": "script-src 'self' https://*.google.com; object-src 'self'"
```

در اینجا استفاده از هر نوع subdomain در سایت گوگل بلامانع اعلام می‌شود.
بنابراین آدرس زیر به background.htm اضافه می‌شود:

```
<script type="text/javascript" src="https://www.google.com/jsapi"></script>
```

استفاده از این Api در rssreader.js

فایل rssreader.js را به background.htm اضافه می‌کنیم و در آن کد زیر را می‌نویسیم:

```
google.load("feeds", "1");
google.setOnLoadCallback(alarManager);
```

آدرسی که ما از گوگل درخواست کردیم فقط مختص خواندن فید نیست؛ تمامی apiهای جاوااسکریپتی در آن قرار دارند و ما تنها نیاز داریم قسمتی از آن لود شود. پس اولین خط از دستور بالا بارگذاری بخش مورد نیاز ما را به عهده دارد. در مورد این دستور [این صفحه](#) را مشاهده کنید.

در خط دوم ما تابع خودمان را به آن معرفی می‌کنیم تا وقتی که گوگل لودش تمام شد این تابع را اجرا کند تا قبل از لود ما از توابع

آن استفاده نکنیم و خطای undefined دریافت نکنیم. تابعی که ما از آن خواستیم اجرا کند alarmManager نام دارد و قرار است یک آلارم و یک سیکل زمانی را ایجاد کرده و در هر دوره، فید را بخواند. کد تابع مدنظر به شرح زیر است:

```
function alarmManager()
{
chrome.storage.local.get(DateContainer.interval,function ( items) {
period_time==items[DateContainer.interval];
chrome.alarms.create('RssInterval', {periodInMinutes: period_time});
});

chrome.alarms.onAlarm.addListener(function (alarm) {
console.log(alarm);
if (alarm.name == 'RssInterval') {

var boolposts,boolpostsComments,boolshares,boolsharesComments;
chrome.storage.local.get([Variables.posts,Variables.postsComments,Variables.shares,Variables.sharesComments],function ( items) {
boolposts=items[Variables.posts];
boolpostsComments=items[Variables.postsComments];
boolshares=items[Variables.shares];
boolsharesComments=items[Variables.sharesComments];

chrome.storage.local.get([DateContainer.posts,DateContainer.postsComments,DateContainer.shares,DateContainer.sharesComments],function ( items) {

var Vposts=new Date(items[DateContainer.posts]);
var VpostsComments=new Date(items[DateContainer.postsComments]);
var Vshares=new Date(items[DateContainer.shares]);
var VsharesComments=new Date(items[DateContainer.sharesComments]);

if(boolposts){var result=RssReader(Links.postUrl,Vposts,DateContainer.posts,Messages.PostsUpdated);}
if(boolpostsComments){var result=RssReader(Links.posts_commentsUrl,VpostsComments,DateContainer.postsComments,Messages.CommentsUpdated);}
if(boolshares){var result=RssReader(Links.sharesUrl,Vshares,DateContainer.shares,Messages.SharesUpdated);}
if(boolsharesComments){var result=RssReader(Links.shares_CommentsUrl,VsharesComments,DateContainer.sharesComments,Messages.SharesCommentsUpdated);}

});
});
}
});
}
```

خطوط اول تابع alarmManager وظیفه‌ی خواندن مقدار interval را که در init.js ذخیره کرده‌ایم، دارند که به طور پیش فرض 10 ذخیره شده است تا تایمر یا آلارم خود را بر اساس آن بسازیم. در خط chrome.alarms.create یک آلارم با نام rssinterval می‌سازد و قرار است هر 10 دقیقه وظایفی که بر دوشش گذاشته می‌شود را اجرا کند (استفاده از api جهت دسترسی به آلارم نیاز به مجوز "alarms" دارد). وظایفش از طریق یک listener که بر روی رویداد chrome.alarms.onAlarm گذاشته شده است مشخص می‌شود. در خط بعدی مشخص می‌شود که این رویداد به خاطر چه آلارمی صدا زده شده است. البته از آنجا که ما یک آلارم داریم، نیاز چندانی به این کد نیست. ولی اگر پروژه شما حداقل دو آلارم داشته باشد نیاز است مشخص شود که کدام آلارم باعث صدا زدن این رویداد شده است. در مرحله بعد مشخص می‌کنیم که کاربر قصد بررسی چه قسمت‌هایی از سایت را داشته است و در تابع callback آن هم تاریخ آخرین تغییرات هر بخش را می‌خوانیم و در متغیری نگه داری می‌کنیم. هر کدام را جداگانه چک کرده و تابع RssReader را برای هر کدام صدا می‌زنیم. این تابع 4 پارامتر دارد:

آدرس فیدی که قرار است از روی آن بخواند

آخرین به روزسانی که از سایت داشته متعلق به چه تاریخی است.

نام کلید ذخیره سازی تاریخ آخرین تغییر سایت که اگر بررسی شد و مشخص شد سایت به روز شده است، تاریخ جدید را روی آن ذخیره کنیم.

در صورتی که سایت به روز شده باشد نیاز است پیامی را برای کاربر نمایش دهیم که این پیام را در اینجا قرار می‌دهیم.

کد تابع rssreader

```
function RssReader(URL,lastupdate,datecontainer,Message) {
    var feed = new google.feeds.Feed(URL);
    feed.setResultFormat(google.feeds.Feed.XML_FORMAT);
    feed.load(function (result) {
if(result!=null)
{
var strRssUpdate = result.xmlDocument.firstChild.firstChild.childNodes[5].textContent;
var RssUpdate=new Date(strRssUpdate);

if(RssUpdate>lastupdate)
{
SaveDateAndShowMessage(datecontainer,strRssUpdate,Message)
}
});
});
}
```

در خط اول فید توسط گوگل خوانده میشود، در خط بعدی ما به گوگل میگوییم که فید خوانده شده را چگونه به ما تحویل دهد که ما قالب xml را خواسته ایم و در خط بعدی اطلاعات را در متغیری به اسم result قرار میدهد که در یک تابع برگشتی آن را در اختیار ما میگذارد. از آن جا که ما قرار است تگ **lastBuildDate** را بخوانیم که پنجمین تگ اولین گره در اولین گره به حساب می آید، خط زیر این دسترسی را برای ما فراهم می کند و چون تگ ما در یک مکان ثابت است با همین تکه کد، دسترسی مستقیمی به آن داریم:

```
var strRssUpdate = result.xmlDocument.firstChild.firstChild.childNodes[5].textContent;
```

مرحله بعد تاریخ را که در قالب رشته ای است، تبدیل به تاریخ کرده و با lastupdate یعنی آخرین تغییر قبلی مقایسه می کنیم و اگر تاریخ برگرفته از فید بزرگتر بود، یعنی سایت به روز شده است و تابع SaveDateAndShowMessage را صدا می زنیم که وظیفه ذخیره سازی تاریخ جدید و ایجاد notification را به عهده دارد و سه پارامتر کلید ذخیره سازی و مقدار آن و پیام را به آن پاس می کنیم.

کد تابع SaveDateAndShowMessage

```
function SaveDateAndShowMessage(DateField,DateValue,Message)
{
var params={
}
params[DateField]=DateValue;

chrome.storage.local.set( params,function(){

var options={
    type: "basic",
    title: Messages.SiteUpdated,
    message: Message,
    imageUrl: "icon.png"
}
chrome.notifications.create("",options,function(){
chrome.notifications.onClicked.addListener(function(){
chrome.tabs.create({'url': WebLinks.Home}, function(tab) {
});
});
});
});
}
```

خطوط اول مربوط به ذخیره تاریخ است و دومین نکته نحوه ی ساخت نوتیفیکیشن است. اجرای یک notification نیاز به مجوز "notifications" دارد که مجوز آن در manifest به شرح زیر است:

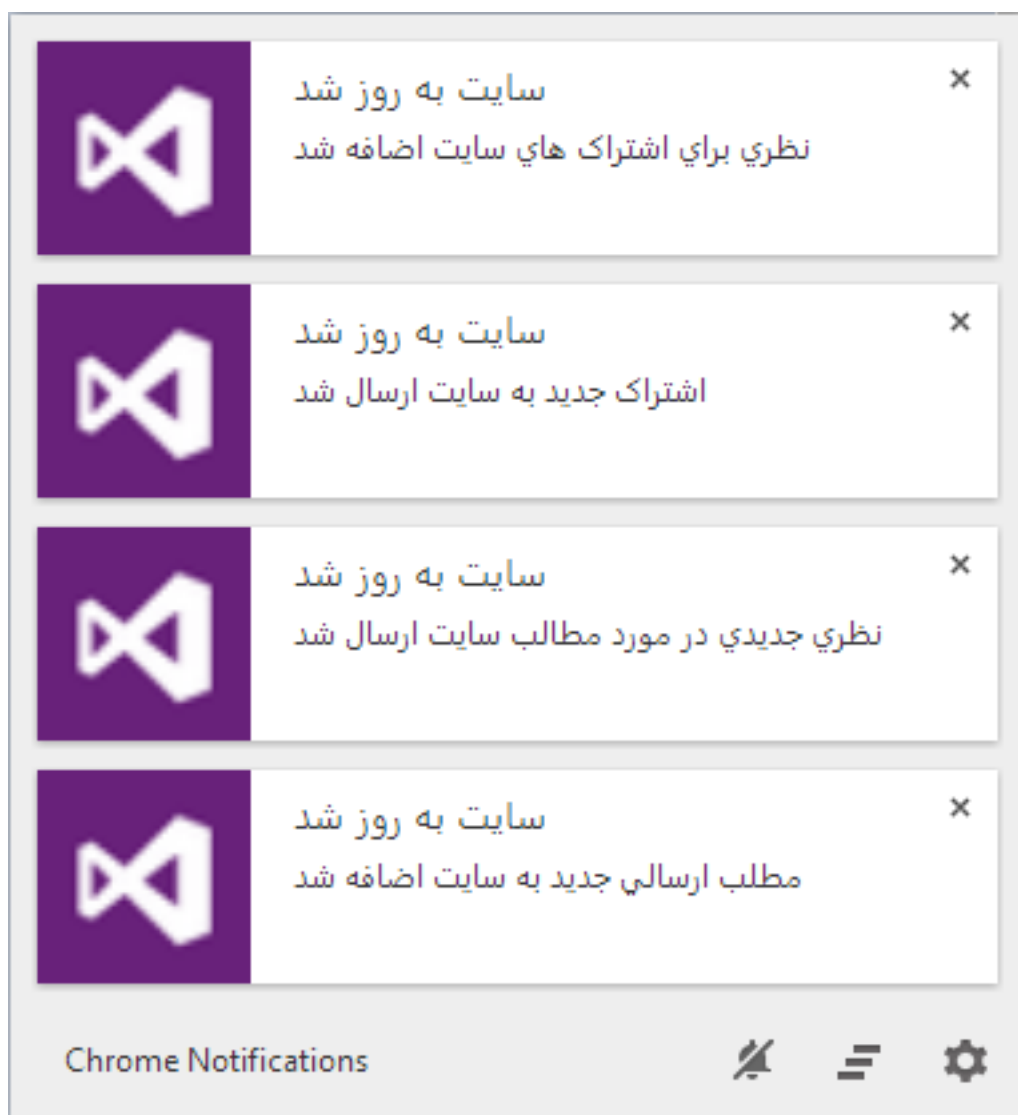
```
"permissions": [
    "storage",
    "tabs",
    "alarms",
    "notifications"
]
```

در خطوط بالا سایر مجوزهایی که در طول این دوره به کار اضافه شده است را هم می‌بینید. برای ساخت نوتیفیکیشن از کد `chrome.notifications.create` استفاده می‌کنیم که پارامتر اول آن کد یکتا یا همان ID جهت ساخت نوتیفیکیشن هست که میتوان خالی گذاشت و دومی تنظیمات ساخت آن است؛ از قبیل عنوان و آیکن و ... که در بالا به اسم `options` معرفی کرده ایم و در آگومان دوم آن را معرفی کرده ایم و آرگومان سوم هم یک تابع `callback` است که نوشتن آن اجباری است. `options` شامل عنوان، پیام، آیکن و نوع `notification` می‌باشد که در اینجا `basic` انتخاب کرده‌ایم. برای دسترسی به دیگر خصوصیت‌های `options` به [اینجا](#) و برای داشتن `notification`‌های زیباتر به عنوان `rich notification` به [اینجا](#) مراجعه کنید. برای اینکه این امکان باشد که کاربر با کلیک روی `notification` به سایت هدایت شود باید در تابع `callback` مربوط به `notifications.create` این کد اضافه گردد که در صورت کلیک یک تب جدید با آدرس سایت ساخته شود:

```
chrome.notifications.create("",options,function(){
chrome.notifications.onClicked.addListener(function(){
chrome.tabs.create({'url': WebLinks.Home}, function(tab) {
});});
});
```

نکته مهم: پیشتر معرفی آیکن به صورت بالا کفایت میکرد ولی بعد از [این باگ](#) کد زیر هم باید جداگانه به `manifest` اضافه شود:

```
"web_accessible_resources": [
  "icon.png"
]
```



خوب؛ کار افزونه تمام شده است ولی اجازه دهید این بار امکانات افزونه را بسط دهیم: من می‌خواهم برای افزونه نیز قسمت تنظیمات داشته باشم. برای دسترسی به options میتوان از قسمت مدیریت افزونه‌ها در مرورگر یا حتی با راست کلیک روی آیکن browser action عمل کرد. در اصل این قسمت برای تنظیمات افزونه است ولی ما به خاطر آموزش و هم اینکه افزونه ما UI خاصی نداشت تنظیمات را از طریق browser action پیاده سازی کردیم و گرنه در صورتی که افزونه شما شامل UI خاصی مثلا نمایش فید مطالب باشد، بهترین مکان تنظیمات، options است. برای تعریف options در manifest.json به روش زیر اقدام کنید:

```
"options_page": "popup.html"
```

همان صفحه popup را در این بخش نشان میدهم و اینبار یک کار اضافه‌تر دیگر که نیاز به آموزش ندارد اضافه کردن input با Type=number است که برای تغییر interval به کار می‌رود و نحوه ذخیره و بازیابی آن را در طول دوره یاد گرفته اید. [جایگزینی صفحات یا Override Pages](#) بعضی صفحات مانند بوک مارک و تاریخچه فعالیت‌ها History و همینطور newtab را می‌توانید جایگزین کنید. البته یک اکستنشن میتواند فقط یکی از صفحات را جایگزین کند. برای تعیین جایگزین در manifest اینگونه عمل می‌کنیم:

```
"chrome_url_overrides": {  
  "newtab": "newtab.htm"  
}
```

ایجاد یک تب اختصاصی در Developer Tools

تکه کدی که باید manifest اضافه شود:

```
"devtools_page": "devtools.htm"
```

شاید فکر کنید کد بالا الان شامل مباحث ui و ... می‌شود و بعد به مرورگر اعمال خواهد شد؛ در صورتی که اینگونه نیست و نیاز دارد چند خط کدی نوشته شود. ولی مسئله اینست که کد بالا تنها صفحات html را پشتیبانی می‌کند و مستقیماً نمی‌تواند فایل js را بخواند. پس صفحه بالا را ساخته و کد زیر را داخلش می‌گذاریم:

```
<script src="devtools.js"></script>
```

فایل devtools.js هم شامل کد زیر می‌شود:

```
chrome.devtools.panels.create(  
  "Dotnettips Updater Tools",  
  "icon.png",  
  "devtoolsui.htm",  
  function(panel) {  
  }  
);
```

خط chrome.devtools.panels.create یک پنل یا همان تب را ساخته و در پارامترهای بالا به ترتیب عنوان، آیکن و صفحه‌ای که باید در آن رندر شود را دریافت می‌کند و پس از ایجاد یک callback اجرا می‌شود. [اطلاعات بیشتر](#)



APIها

برای دیدن لیست کاملی از APIها می‌توانید به [مستندات](#) آن رجوع کنید و این مورد را به یاد داشته باشید که ممکن است بعضی apiها در بعضی موارد پاسخ ندهند. به عنوان مثال در content scripts نمی‌توانید به chrome.devtools.panels دسترسی داشته باشید یا اینکه در DeveloperTools دسترسی به DOM میسر نیست. پس این مورد را به خاطر داشته باشید. همچنین بعضی apiها از نسخه‌ی خاصی به بعد اضافه شده‌اند مثلاً همین مثال قبلی devtools از نسخه 18 به بعد اضافه شده است و به این معنی است با خیال راحت می‌توانید از آن استفاده کنید. یا آلارم‌ها از نسخه 22 به بعد اضافه شده‌اند. البته خوشبختانه امروزه با دسترسی آسانتر به اینترنت و آپدیت خودکار مرورگرها این مشکلات دیگر آن چنان رخ نمی‌دهند.

Messaging

همانطور که در بالا اشاره شد شما نمی‌توانید بعضی از apiها را در بعضی جاها استفاده کنید. برای حل این مشکل می‌توان از messaging استفاده کرد که دو نوع تبادلات پیغامی داریم:
One-Time Requests یا درخواست‌های تک مرتبه‌ای
Long-Lived Connections یا اتصالات بلند مدت یا مصر

درخواست‌های تک مرتبه ای

این درخواست‌ها همانطور که از نامش پیداست تنها یک مرتبه رخ می‌دهد؛ درخواست را ارسال کرده و منتظر پاسخ می‌ماند. به عنوان مثال به کد زیر که در content script است دقت کنید:

```
window.addEventListener("load", function() {
  chrome.extension.sendMessage({
    type: "dom-loaded",
    data: {
      myProperty: "value"
    }
  });
}, true);
```

کد بالا یک ارسال کننده پیام است. موقعی که سایتی باز می‌شود، یک کلید با مقدارش را ارسال می‌کند و کد زیر در background گوش می‌ایستد تا اگر درخواستی آمد آن را دریافت کند:

```
chrome.extension.onMessage.addListener(function(request, sender, sendResponse) {
  switch(request.type) {
    case "dom-loaded":
      alert(request.data.myProperty);
      break;
  }
  return true;
});
```

اتصالات بلند مدت یا مصر

اگر نیاز به یک کانال ارتباطی مصر و همیشگی دارید کدها را به شکل زیر تغییر دهید contentscripts

```
var port = chrome.runtime.connect({name: "my-channel"});
port.postMessage({myProperty: "value"});
port.onMessage.addListener(function(msg) {
  // do some stuff here
});
```

background

```
chrome.runtime.onConnect.addListener(function(port) {
  if(port.name == "my-channel"){
    port.onMessage.addListener(function(msg) {
      // do some stuff here
    });
  }
});
```

نمونه کد نمونه کدهایی که در سایت گوگل موجود هست می‌توانند کمک بسیاری خوبی باشند ولی اینگونه که پیداست اکثر مثال‌ها مربوط به نسخه‌ی یک manifest است که دیگر توسط مرورگرها پشتیبانی نمی‌شوند و مشکلاتی چون اسکریپت inline و CSP که در بالا اشاره کردیم را دارند و گوگل کدها را به روز نکرده است.

دیباگ کردن و پک کردن فایل‌ها برای تبدیل به فایل افزونه Debugging and packing

برای دیباگ کردن کدها می‌توان از دو نمونه console.log و alert برای گرفتن خروجی استفاده کرد و همچنین ابزار [Chrome Apps](#) [Extensions Developer Tool](#) هم نسبتاً امکانات خوبی دارد که البته می‌توان از آن برای پک کردن اکستنشن نهایی هم استفاده کرد. برای پک کردن روی گزینه pack کلیک کرده و در کادر باز شده گزینه‌ی pack را بزنید. برای شما دو نوع فایل را در مسیر والد دایرکتوری extension نوشته شده درست خواهد کرد که یکی پسوند crx دارد که می‌شود همان فایل نهایی افزونه و دیگری هم پسوند pem دارد که یک کلید اختصاصی است و باید برای آپدیت‌های آینده افزونه آن را نگاه دارید. در صورتی که افزونه را تغییر دادید و خواستید آن را به روز رسانی کنید موقعی که اولین گزینه pack را می‌زنید و صفحه باز می‌شود قبل از اینکه دومین گزینه pack را بزنید، از شما می‌خواهد اگر دارید عملیات به روز رسانی را انجام می‌دهید، کلید اختصاصی آن را وارد نمایید و بعد از آن گزینه pack را بزنید:



Dotnettips Updater 1.0

This extension keeps you updated on current activities on dotnettips.info

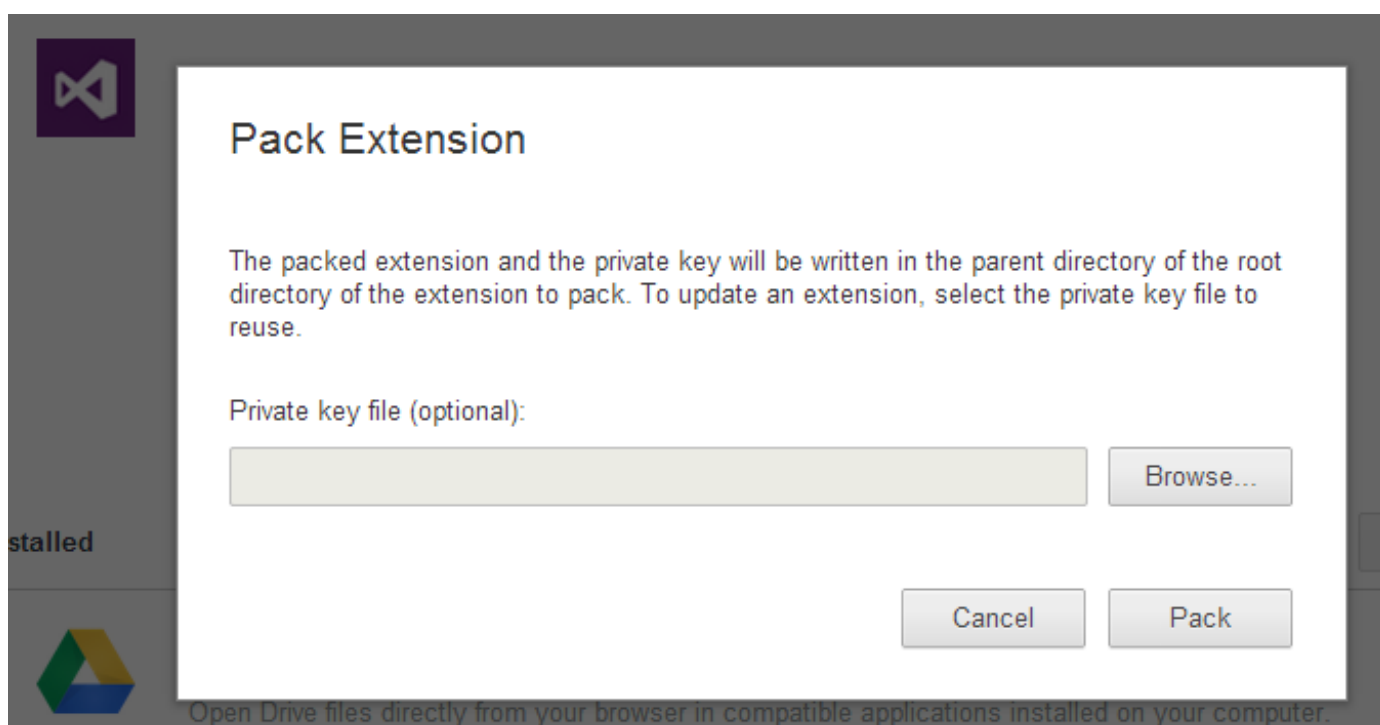
ID: eplfgnfdfogebcbdakakijnlkmbhko

Loaded from: C:\Users\aym\Desktop\chrome Ext

☒ Enabled ☐ Allow in incognito

Inspect views: background.htm

Reload Permissions Behavior **Pack** Uninstall



آپلود نهایی کار در Google web store

برای آپلود نهایی کار به [google web store](https://chrome.google.com/webstore/developer/dashboard) که در آن تمامی برنامه‌ها و افزونه‌های کروم قرار دارند بروید. سمت راست آیکن تنظیمات را بزنید و گزینه developer dashboard را انتخاب کنید تا صفحه‌ی آپلود کار برای شما باز شود. دایرکتوری محتویات اکستنشن را zip کرده و آپلود نمایید. توجه داشته باشید که محتویات و سورس خود را باید آپلود کنید نه فایل crx را. بعد از آپلود موفقیت آمیز، صفحه‌ای ظاهر می‌شود که از شما آیکن افزونه را در اندازه 128 پیکسل می‌خواهد بعلاوه توضیحاتی در مورد افزونه، قیمت گذاری که به طور پیش فرض به صورت رایگان تنظیم شده است، لینک وب سایت مرتبط، لینک محل پرسش و پاسخ برای افزونه، اگر لینک یوتیوبی در مورد افزونه دارید، یک شات تصویری از افزونه و همینطور چند تصویر برای اسلایدشو سازی که در همان صفحه استاندارد آن‌ها را توضیح می‌دهد و در نهایت گزینه‌ی جالب‌تر هم اینکه اکستنشن شما برای چه مناطقی تهیه شده است که متاسفانه ایران را ندیدم که می‌توان همه موارد را انتخاب کرد. به خصوص در مورد ایران که آی پی‌ها هم صحیح نیست، انتخاب ایران چنان تاثیری ندارد و در نهایت گزینه‌ی publish را می‌زنید که متاسفانه بعد از این صفحه درخواست می‌کند برای اولین بار باید 5 دلار آمریکا پرداخت شود که برای بسیاری از ما این گزینه ممکن نیست.

سورس پروژه را می‌توانید از [اینجا](#) ببینید و خود افزونه را از [اینجا](#) دریافت کنید.

در دو مقاله پیشین [^](#) ، [^](#) به بررسی نوشتن افزونه در مرورگر کروم پرداختیم و اینبار قصد داریم همان پروژه را برای فایرفاکس پیاده کنیم. پس در مورد کدهای تکراری توضیحی داده نخواهد شد و برای فهم آن می‌توانید به دو مقاله قبلی رجوع کنید. همه‌ی ما فایرفاکس را به خوبی می‌شناسیم. اولین باری که این مرورگر آمد سرو صدای زیادی به پا کرد و بازار وسیعی از مرورگرها را که در چنگ IE بود، به دست آورد. این سرو صدا بیشتر به خاطر امنیت و کارایی بالای این مرورگر، استفاده از آخرین فناوری‌های تحت وب و دوست داشتنی برای طراحان وب بود. همچنین یکی دیگر از مهمترین ویژگی‌های آن، امکان سفارشی سازی آن با افزونه‌ها یا extensions یا addon بود که این ویژگی در طول این سال‌ها تغییرات زیادی به خود دیده است. در مورد افزونه نویسی برای فایرفاکس در سطح نت مطالب زیادی وجود دارند که همین پیشرفت‌های اخیر در مورد افزونه‌ها باعث شده خیلی از این مطالب به روز نباشند. اگر در مقاله پیشین فکر می‌کنید که کروم چقدر در نوشتن افزونه جذابیت دارد و امکانات خوبی را در اختیار شما می‌گذارد، الان دیگر وقت آن است که نظر خودتان را عوض کنید و فایرفاکس را نه تنها یک سرو گردن بلکه بیشتر از این حرف‌ها بالاتر بدانید.

شرکت موزیلا برای قدرتمندی و راحتی کار طراحان یک sdk طراحی کرده است و شما با استفاده از کدهای موجود در این sdk قادرید کارهای زیادی را انجام دهید. برای نصب این sdk باید پیش نیازهایی بر روی سیستم شما نصب باشد: نصب پایتون 2.5 یا 2.6 یا 2.7 که فعلا در سایت آن، نسخه‌ی 2.7 در دسترس هست. توجه داشته باشید که هنوز برای نسخه‌ی 3 پایتون پشتیبانی صورت نگرفته است.

آخرین نسخه‌ی sdk را هم می‌توانید از این [آدرس](#) به صورت zip و یا از این [آدرس](#) به صورت tar دانلود کنید و در صورتیکه دوست دارید به سورس آن دسترسی داشته باشید یا اینکه از سورس‌های مشارکت شده یا غیر رسمی استفاده کنید، از این [صفحه](#) آن را دریافت کنید.

بعد از دانلود sdk به شاخه‌ی bin رفته و فایل activate.bat را اجرا کنید. موقعی که فایل activate اجرا شود، باید چنین چیزی دیده شود:

```
(C:\Users\aym\Downloads\addon-sdk-1.17) C:\Users\aym\Downloads\addon-sdk-1.17\bin>
```

برای سیستم‌های عامل Linux, FreeBSD, OS X دستور زیر را وارد کنید:
اگر یک کاربر پوسته‌ی bash هستید کلمه زیر را در کنسول برای اجرای activate بزنید:

```
source bin/activate
```

اگر کاربر پوسته‌ی بش نیستید:

```
bash bin/activate
```

نهایتا باید کنسول به شکل زیر در آید یا شبیه آن:

```
(addon-sdk)~/mozilla/addon-sdk >
```

بعد از اینکه به کنسول آن وارد شدید، کلمه cfx را در آن تایپ کنید تا راهنمای دستورات و سوییچ‌های آن‌ها نمایش داده شوند. از این ابزار میتوان برای راه اندازی فایرفاکس و اجرای افزونه بر روی آن، پکیج کردن افزونه، دیدن مستندات و [آزمون‌های واحد](#) استفاده کرد.

آغاز به کار

برای شروع، فایل‌های زیادی باید ساخته شوند، ولی نگران نباشید cfx این کار را برای شما خواهد کرد. دستورات زیر را جهت

ساخت یک پروژه خالی اجرا کنید:

```
mkdir fxaddon
cd fxaddon
cfx init
```

یک پوشه را در مسیری که کنسول بالا اشاره میکرد، ساختیم و وارد آن شدیم و با دستور `cfx init` دستور ساخت یک پروژهی خالی را دادیم و باید بعد از این دستور، یک خروجی مشابه زیر نشان بدهد:

```
* lib directory created
* data directory created
* test directory created
* doc directory created
* README.md written
* package.json written
* test/test-main.js written
* lib/main.js written
* doc/main.md written
Your sample add-on is now ready for testing:
try "cfx test" and then "cfx run". Have fun!"
```

در این پوشه یک فایل به اسم [package.json](#) هم وجود دارد که اطلاعات زیر داخلش هست:

```
{
  "name": "fxaddon",
  "title": "fxaddon",
  "id": "jid1-QfyqpNby9lTlcQ",
  "description": "a basic add-on",
  "author": "",
  "license": "MPL 2.0",
  "version": "0.1"
}
```

این اطلاعات شامل نام و عنوان افزونه، توضیحی کوتاه در مورد آن، نویسندهی افزونه، ورژن افزونه و ... است. این فایل دقیقا معادل `manifest.json` در کروم است. در افزونه نویسی‌های قدیم این فایل `install.rdf` نام داشت و بر پایهی فرمت `rdf` بود. ولی در حال حاضر با تغییرات زیادی که افزونه نویسی در فایرفاکس کرده‌است، الان این فایل بر پایه یا فرمت `json` است. اطلاعات `package` را به شرح زیر تغییر می‌دهیم:

```
{
  "name": "dotnettips",
  "title": ".net Tips Updater",
  "id": "jid1-QfyqpNby9lTlcQ",
  "description": "This extension keeps you updated on current activities on dotnettips.info",
  "author": "yeganehaym@gmail.com",
  "license": "MPL 2.0",
  "version": "0.1"
}
```

رابطه‌های کاربری Action Button و Toggle Button فایل `main.js` را در دایرکتوری `lib` باز کنید:

موقعی که در کروم افزونه می‌نوشتیم امکانی به اسم `browser action` داشتیم که در اینجا با نام `action button` شناخته می‌شود. در اینجا باید کدها را `require` کرد، همان کاری در خیلی از زبان‌ها مثلا مثل سی برای صدا زدن سرآیندها می‌کنید. مثلا برای `action button` اینگونه است:

```
var button= require('sdk/ui/button/action');
```

نحوه‌ی استفاده هم بدین صورت است:

```
buttons.ActionButton({...});
```

که در بین {} خصوصیات دکمه‌ی مورد نظر نوشته می‌شود. ولی من بیشتر دوست دارم از شیء دیگری استفاده کنم. به همین جهت ما از یک مدل دیگر button که به اسم toggle button شناخته می‌شود، استفاده می‌کنیم. از آن جا که این button دارای دو حالت انتخاب (حالت فشرده شده) و غیر انتخاب (معمولی و آماده فشرده شدن توسط کلیک کاربر) است، بهترین انتخاب هست.



کد زیر یک toggle button را برای فایرفاکس می‌سازد که با کلیک بر روی آن، صفحه‌ی popup.htm به عنوان یک پنل روی آن رندر می‌شود:

```
var tgbutton = require('sdk/ui/button/toggle');
var panels = require("sdk/panel");
var self = require("sdk/self");

var button = tgbutton.ToggleButton({
  id: "updaterui",
  label: ".Net Updater",
  icon: {
    "16": "./icon-16.png",
    "32": "./icon-32.png",
    "64": "./icon-64.png"
  },
  onChange: handleChange
});

var panel = panels.Panel({
  contentURL: self.data.url("./popup.html"),
  onHide: handleHide
});

function handleChange(state) {
  if (state.checked) {
    panel.show({
      position: button
    });
  }
}

function handleHide() {
  button.state('window', {checked: false});
}
```

در سه خط اول، فایل‌هایی را که نیاز است Required شوند، می‌نویسیم و در یک متغیر ذخیره می‌کنیم. اگر در متغیر نریزیم مجبور هستیم همیشه هر کدی را به جای نوشتن عبارت زیر:

```
tgbutton.ToggleButton
```

به صورت زیر بنویسیم:

```
require('sdk/ui/button/toggle').ToggleButton
```

که اصلا کار جالبی نیست. اگر مسیرهای نوشته شده را از مبدا فایل zip که اکستراکت کرده‌اید، در دایرکتوری sdk در شاخه lib بررسی کنید، با دیگر موجودیت‌های sdk آشنا خواهید شد.

در خط بعدی به تعریف یک شیء از نوع toggle button به اسم button می‌پردازیم و خصوصیتی که به این دکمه داده ایم، مانند یک کد شناسایی، یک برچسب که به عنوان tooltip نمایش داده خواهد شد و آیکن‌هایی در اندازه‌های مختلف که در هرجایی کاربر آن دکمه را قرار داد، در اندازه‌ی مناسب باشد و نهایتا به تعریف یک رویداد می‌پردازیم. تابع handlechange زمانی صدا زده می‌شود که در وضعیت دکمه‌ی ایجاد شده تغییری حاصل شود. در خط بعدی شیء panel را به صورت global می‌سازیم. شیء self دسترسی ما را به اجزا یا فایل‌های افزونه خودمان فراهم می‌کند که در اینجا دسترسی ما به فایل html در شاخه‌ی data میسر شده است و مقدار مورد نظر را در contentURL قرار می‌دهد. نهایتا هم برای رویداد onhide تابعی را در نظر می‌گیریم تا موقعی که پنجره بسته شد بتوانیم وضعیت toggle button را به حالت قبلی بازگردانیم و حالت فشرده نباشد. چرا که این دکمه تنها با کلیک ماوس به حالت فشرده و حالت معمولی سوییچ میکند. پس اگر کاربر با کلیک بر روی صفحه‌ی مرورگر پنجره را ببندد، دکمه در همان وضعیت فشرده باقی می‌ماند.

همانطور که گفتیم تابع handlechange موقعی رخ می‌دهد که در وضعیت دکمه، تغییری رخ دهد و نمیدانیم که این وضعیت فشرده شدن دکمه هست یا از حالت فشرده خارج شده است. پس با استفاده از ویژگی checked بررسی می‌کنیم که آیا دکمه‌ای فشرده شده یا خیر؛ اگر برابر true بود یعنی کاربر روی دکمه، کلیک کرده و دکمه به حالت فشرده رفته، پس ما هم پنل را به آن نشان می‌دهیم و خصوصیات دلخواهی را برای مشخص کردن وضعیت پنل نمایشی به آن پاس می‌کنیم. [خصوصیت یا پارامترهای زیادی](#) را می‌توان در حین ساخت پنل برای آن ارسال کرد. با استفاده از خصوصیت position محل نمایش پنجره را مشخص می‌کنیم. در صورتی که ذکر نشود پنجره در وسط مرورگر ظاهر خواهد شد.

تابع onhide زمانی رخ می‌دهد که به هر دلیلی پنجره بسته شده باشد که در بالا یک نمونه‌ی آن را عرض کردیم. ولی اتفاقی که می‌افتد، وضعیت تابع را با متد state تغییر می‌دهیم و خصوصیت checked آن را false می‌کنیم. بجای پارامتر اولی، دو گزینه را میتوان نوشت؛ یکی window و دیگری tab است. اگر شما گزینه tab را جایگزین کنید، اگر در یک تب دکمه به حالت فشرده برود و به تب دیگر بروید و باعث بسته شدن پنجره بشوید، دکمه تنها در تبی که فعال است به حالت قبلی باز می‌گردد و تب اولی همچنان حالت خود را حفظ خواهد کرد پس می‌نویسیم window تا این عمل در کل پنجره اعمال شود.

Context Menus

برای ساخت منوی کانتکست از کد زیر استفاده می‌کنیم:

```
var contextMenu = require("sdk/context-menu");

var home = contextMenu.Item({
  label: "صفحه اصلی",
  data: "http://www.dotnettips.info/"
});
var postsarchive = contextMenu.Item({
  label: "مطالب سایت",
  data: "http://www.dotnettips.info/postsarchive"
});

var menuItem = contextMenu.Menu({
  label: "Open .Net Tips",
  context: contextMenu.PageContext(),
  items: [home, postsarchive],
  image: self.data.url("icon-16.png"),
  contentScript: 'self.on("click", function (node, data) {' +
    '  window.location.href = data;' +
    '});'
```

این منو هم مثل کروم دو زیر منو دارد که یکی برای باز کردن صفحه‌ی اصلی و دیگری برای باز کردن صفحه‌ی مطالب است. هر کدام یک برچسب برای نمایش متن دارند و یکی هم دیتا که برای نگهداری آدرس است. در خط بعدی منوی پدر یا والد ساخته می‌شود که با خصوصیت `items`، زیر منوهایش را اضافه می‌کنیم و با خصوصیت `image`، تصویری را در پوشه‌ی دیتا به آن معرفی می‌کنیم که اندازه‌ی آن 16 پیکسل است و دومی هم خصوصیت `context` است که مشخص می‌کند این گزینه در چه مواردی بر روی `context menu` نمایش داده شود. الان روی همه چیزی نمایش داده می‌شود. اگر گزینه، `SelectionContext` باشد، موقعی که متنی انتخاب شده باشد، نمایش می‌یابد. اگر `SelectorContext` باشد، خود شما مشخص می‌کنید بر روی چه مواردی نمایش یابد؛ مثلاً عکس یا تگ p یا هر چیز دیگری، کد زیر باعث می‌شود فقط روی عکس نمایش یابد:

```
SelectorContext("img")
```

کد زیر هم روی عکس و هم روی لینکی که href داشته باشد:

```
SelectorContext("img,a[href]")
```

موارد دیگری هم وجود دارند که می‌توانید مطالب بیشتری را در مورد آن‌ها در [اینجا](#) مطالعه کنید. آخرین خصوصیت باقی مانده، `content script` است که می‌توانید با استفاده از جاوااسکریپت برای آن کد بنویسید. موقعی که برای آن رویداد کلیک رخ داد، مشخص شود تابعی را صدا می‌زند با دو آرگومان؛ [گروه](#) ای که انتخاب شده و داده‌ای که به همراه دارد که آدرس سایت است و آن را در نوار آدرس درج می‌کند.

آن منوهای که با متد `item` ایجاد شده‌اند منوهای هستند که با کلیک کاربر اجرا می‌شوند؛ ولی والدی که با متد `menu` ایجاد شده است، برای منویی است که زیر منو دارد و خودش لزومی به اجرای کد ندارد. پس اگر منویی می‌سازید که زیرمنو ندارد و خودش قرار است کاری را انجام دهد، به صورت همان `item` بنویسید که پایین‌تر نمونه‌ی آن را خواهید دید. الان مشکلی که ایجاد می‌شود این است که موقعی که سایت را باز می‌کند، در همان تبی رخ می‌دهد که فعال است و اگر کاربر بر روی صفحه‌ی خاصی باشد، آن صفحه به سمت سایت مقصد رفته و سایت فعلی از دست می‌رود. روش صحیح‌تر اینست که تبی جدید بار شود و آدرس مقصد در آن نمایش یابد. پس باید از روشی استفاده کنیم که رویداد کلیک توسط کد خود افزونه مدیریت شود، تا با استفاده از شیء `tab`، یک تب جدید با آدرسی جدید ایجاد کنیم. پس کد را با کمی تغییر می‌نویسیم:

```
var tabs = require("sdk/tabs");
var menuItem = contextMenu.Menu({
  label: "Open .Net Tips",
  context: contextMenu.PageContext(),
  items: [home, postsarchive],
  image: self.data.url("icon-16.png"),
  contentScript: 'self.on("click", function (node, data) {' +
    '  self.postMessage(data);' +
    '});',
  onMessage: function (data) {
    tabs.open(data);
  }
});
```

با استفاده از `postmessage`، هر پارامتری را که بخواهیم ارسال می‌کنیم و بعد با استفاده از رویداد `onMessage`، داده‌ها را خوانده و کد خود را روی آن‌ها اجرا می‌کنیم. بگذارید کد زیر را هم جهت سرچ مطالب بر روی سایت پیاده کنیم:

```
var Url="http://www.dotnettips.info/search?term=";
var searchMenu = contextMenu.Item({
  label: "search for",
  context: [contextMenu.PredicateContext(checkText),contextMenu.SelectionContext()],
  image: self.data.url("icon-16.png"),
  contentScript: 'self.on("click", function () {' +
    '  var text = window.getSelection().toString();' +
    '  if (text.length > 20)' +
    '    text = text.substr(0, 20);' +
    '  self.postMessage(text);' +
    '});',
  onMessage: function (data) {
```



```

    tabs.open(Url+data);
  }
});
function checkText(data) {
    if(data.selectionText === null)
        return false;

    console.log('selectionText: ' + data.selectionText);

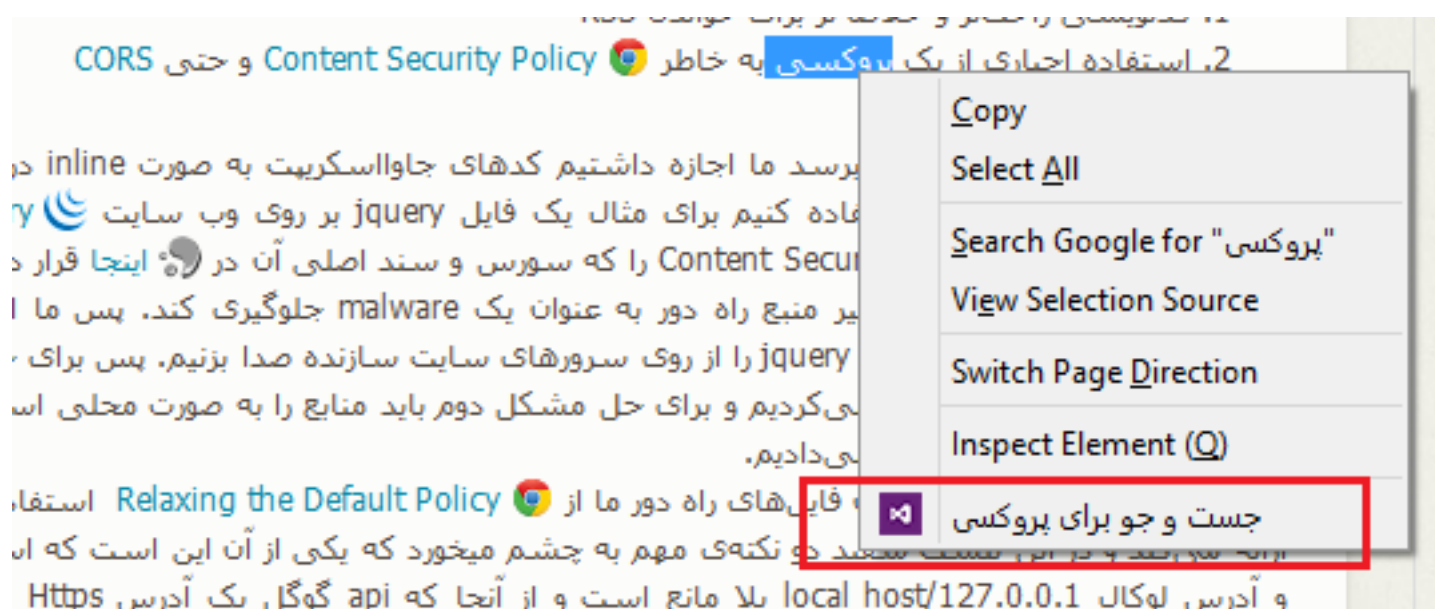
    //handle showing or hiding of menu items based on the text content.
    menuItemToggle(data.selectionText);

    return true;
};
function menuItemToggle(text){
    var searchText="جست و جو برای";
    searchMenu.label=searchText+text;
};

```

در ساخت این منو، ما از ContextSelection استفاده کرده ایم. بدین معنی که موقعی که چیزی روی صفحه انتخاب شد، این منو ظاهر شود و گزینه‌ی دیگری که در کنارش هست، گزینه contextMenu.PredicateContext وظیفه دارد تابعی که به عنوان آرگومان به آن دادیم را موقعی که منو کانتکست ایجاد شد، صدا بزند و اینگونه میتوانیم بر حسب اطلاعات کانتکست، منوی خود را ویرایش کنیم. مثلاً من دوست دارم موقعی که متنی انتخاب می‌شود و راست کلیک می‌کنم گزینه‌ی "جست و جو برای..." نمایش داده شود و به جای ... کلمه‌ی انتخاب شده نمایش یابد. به شکل زیر دقت کنید. این چیزی است که ما قرار است ایجاد کنیم:

در کل موقع ایجاد منو تابع checkText اجرا شده و متن انتخابی را خوانده به عنوان یک آرگومان برای تابع menuItemToggle ارسال می‌کند و به رشته "جست و جو برای" می‌چسباند. در خود پارامترهای آیتم اصلی، گزینه content scrip، با استفاده از جاوااسکریپت، متن انتخاب شده را دریافت کرده و با استفاده از متد postmessage برای تابع onMessage ارسال کرده و با ساخت یک تب و چسباندن عبارت به آدرس جست و جو سایت، کاربر را به صفحه مورد نظر هدایت کرده و عمل جست و جو در سایت انجام می‌گیرد.



در قسمت آینده موارد بیشتری را در مورد افزونه نویسی در فایرفاکس بررسی خواهیم کرد و افزونه را تکمیل خواهیم کرد

نظرات خوانندگان

نویسنده: مهدی رو
تاریخ: ۲۰:۴۶ ۱۳۹۴/۰۳/۲۸

با سلام؛ لطفا بگید چه طور قدم به قدم این کد هایی رو که مینویسیم اجرا کنیم؟ مرسی

نویسنده: علی یگانه مقدم
تاریخ: ۲۲:۱۷ ۱۳۹۴/۰۳/۲۸

همینطور که در بالا گفتیم:
بعد از اینکه به کنسول آن وارد شدید، کلمه `cfx` را در آن تایپ کنید تا راهنمای دستورات و سوییچ های آن ها نمایش داده شوند. از این ابزار میتوان برای راه اندازی فایرفاکس و اجرای افزونه بر روی آن، پکیج کردن افزونه، دیدن مستندات و [آزمون های واحد](#) استفاده کرد.

از دستور زیر در مسیر دایرکتوری افزونه برای اجرا و تست افزونه استفاده می کنیم:

```
cfx run
```

انتهای مطلب [قسمت دوم](#) برای اجرا و بسته بندی افزونه این مورد رو توضیح دادیم.

در مقاله [پیشین](#) ، افزونه نویسی برای فایرفاکس را آغاز و مسائل مربوط به رابط‌های کاربری را بررسی کردیم. در این قسمت که قسمت پایانی افزونه نویسی برای فایرفاکس است، به مباحث پردازشی و دیگر خصوصیت‌ها می‌پردازیم.

اولین موردی که باید برای برنامه‌ی ما در نظر گرفت، ذخیره و بازیابی مقادیر است که باید روی پنجره‌ی popup.html اعمال گردد و همچنین مقداردهی مقادیر پیش فرض برنامه بعد از نصب افزونه اعمال شود. برای ذخیره‌ی مقادیر، طبق نوشته موجود در راهنمای موزیلا، از روش زیر بهره برده و می‌توان مقادیر زیر را به راحتی در آن‌ها ذخیره کرد:

```
var ss = require("sdk/simple-storage");
ss.storage.myArray = [1, 1, 2, 3, 5, 8, 13];
ss.storage.myBoolean = true;
ss.storage.myNull = null;
ss.storage.myNumber = 3.1337;
ss.storage.myObject = { a: "foo", b: { c: true }, d: null };
ss.storage.myString = "O frabjous day!";
```

برای خواندن موارد ذخیره شده هم که مشخصاً نوشتن اسم property کفایت می‌کند و برای حذف مقادیر نیز به راحتی از عبارت delete در جلوی پراپرتی استفاده کنید:

```
delete ss.storage.value;
```

برای ذخیره مقادیر پیش فرض اولین، کاری که می‌کنیم اسم متغیرها را چک می‌کنیم. اگر مخالف null بود، یعنی قبلاً ست شده‌اند؛ ولی اگر null شد، عمل ذخیره سازی اولیه را انجام می‌دهیم:

```
if (!ss.storage.Variables)
{
    ss.storage.Variables=[];
    ss.storage.Variables.push(true);
    ss.storage.Variables.push(false);
    ss.storage.Variables.push(false);
    ss.storage.Variables.push(false);
}

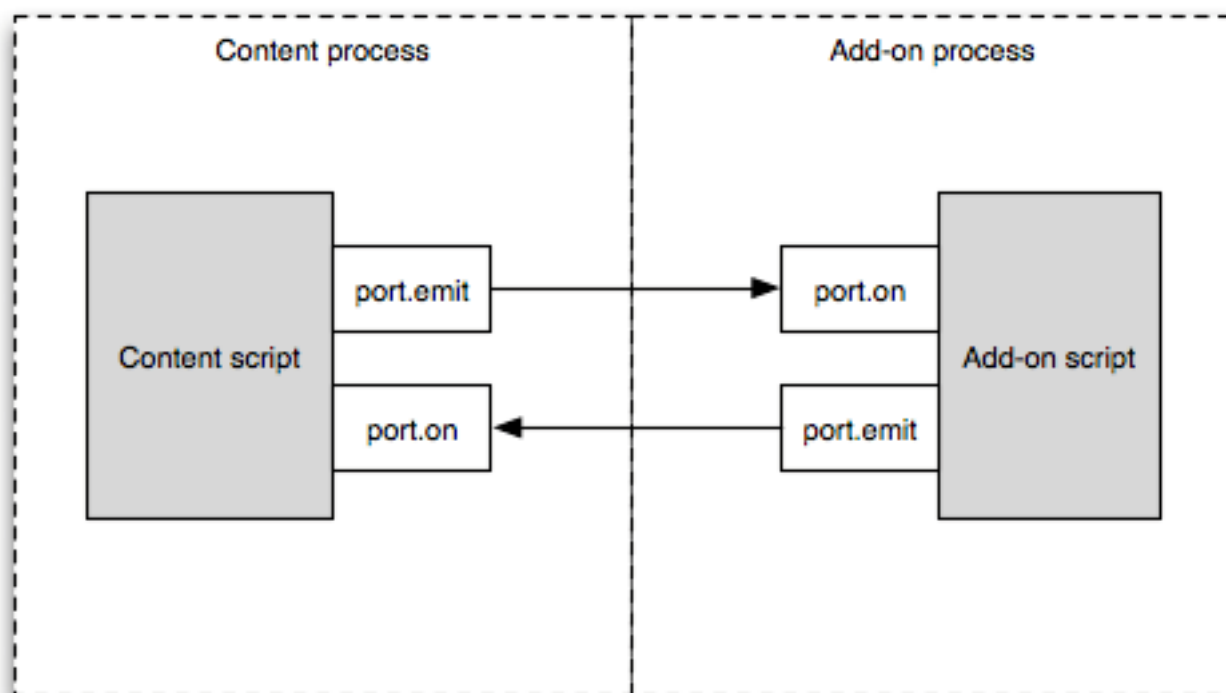
if (!ss.storage.interval)
ss.storage.interval=1;

if (!ss.storage.DateVariables)
{
    var now=String(new Date());
    ss.storage.DateVariables=[];
    ss.storage.DateVariables.push(now);
    ss.storage.DateVariables.push(now);
    ss.storage.DateVariables.push(now);
    ss.storage.DateVariables.push(now);
}
```

برای ذخیره مقادیر popup.html، به طور مستقیم نمی‌توانیم از کدهای بالا در جاوااسکریپت استفاده کنیم. مجبور هستیم که یک پل ارتباطی بین فایل main.js و فایل جاوااسکریپت داشته باشیم. در مقاله پیشین در مورد postmessage که ارتباطی از/به محتوا یا فایل جاوااسکریپت به/از main.js برقرار می‌کرد، صحبت کردیم و در این قسمت راه حل بهتری را مورد استفاده قرار می‌دهیم.

برای ایجاد چنین ارتباطی، آن هم به صورت دو طرفه از [port](#) استفاده می‌کنیم. دستور پورت در اکثر اشیایی که ایجاد می‌کنید وجود دارد ولی باز هم همیشه قبل از استفاده، از مستندات موزیلا حتماً استفاده کنید تا مطمئن شوید دسترسی به شیء پورت در همه‌ی اشیاء وجود دارد. ولی به صورت کلی تا آنجایی که من دیدم، در همه‌ی اشیاء قرار دارد. از کد port.emit برای ارسال مقادیر به سمت فایل اسکریپت یا حتی بالعکس مورد استفاده قرار می‌گیرد و port.on هم یک شنونده برای آن است. شکل زیر به خوبی این

مبحث را نشان می‌دهد.



addon process در شکل بالا همان فایل main.js هست که کد اصلی addon داخل آن است و content process نیز محتوای اسکریپت است و حالا میتواند با استفاده از خاصیت contentscrip که به صورت رشته ای اعمال شده باشد یا اینکه با استفاده از خاصیت contentScriptFile، در یک یا چند فایل js استفاده نماید:

```
contentScriptFile: self.data.url("jquery.min.js")
contentScriptFile: [self.data.url("jquery.min.js"),self.data.url("const.js"),self.data.url("popup.js")]
```

از شیء port به صورت عملی استفاده می‌کنیم. کد main.js را به صورت زیر تغییر دادیم:

```
function handleChange(state) {
  if (state.checked) {
    panel.show({
      position: button
    });

    var v1=[],v2;
    if (ss.storage.Variables)
      v1=ss.storage.Variables;

    if (ss.storage.interval)
      v2=ss.storage.interval;

    panel.port.emit("vars",v1,v2);
  }
  panel.port.on("vars", function (vars,interval) {
    ss.storage.Variables=vars;
    ss.storage.interval=interval;
  });
}
```

در شماره پیشین گفتیم که رویداد handleChange وظیفه نمایش پنل را دارد، ولی الان به غیر آن چند سطر، کد دیگری را هم اضافه کردیم تا موقعی که پنل باز می‌شود، تنظیمات قبلی را که ذخیره کرده‌ایم روی صفحه نمایش داده شوند. با استفاده از شیء

port.emit محتوای دریافت شده را به سمت فایل اسکریپت ارسال می‌کنیم تا تنظیمات ذخیره شده را برای نمایش اعمال کند. پارامتر اول رشته vars نام پیام رسان شما خواهد بود و در فایل مقصد هم تنها به پیامی با این نام گوش داده خواهد شد. این خصوصیت زمانی سودمندی خود را نشان می‌دهد که بخواهید در زمینه‌های مختلف، از چندین پیام رسان به سمت یک مقصد استفاده کنید. شیء panel.port.on هم برای گوش دادن به متغیرهایی است که از آن سمت برای ما ارسال می‌شود و از آن برای ذخیره‌ی مواردی استفاده می‌گردد که کاربر از آن سمت برای ما ارسال می‌کند. پس ما در این مرحله، یک ارتباطه کاملاً دو طرفه داریم.

کد فایل popup.js که به صورت تگ script در popup.html معرفی شده است:

```
$(document).ready(function () {
    addon.port.on("vars", function(vars,interval) {
        if (vars)
        {
            $("#chkarticles").attr("checked", vars[0]);
            $("#chkarticlescomments").attr("checked", vars[1]);
            $("#chkshares").attr("checked", vars[2]);
            $("#chksharescomments").attr("checked", vars[3]);
        }

        $("#interval").val(interval);
    });

    $("#btnsave").click(function() {
        var Vposts = $("#chkarticles").is(':checked');
        var VpostsComments = $("#chkarticlescomments").is(':checked');
        var Vshares = $("#chkshares").is(':checked');
        var VsharesComments = $("#chksharescomments").is(':checked');
        var Vinterval = $("#interval").val() ;
        var Variables=[];
        Variables[0]=Vposts;
        Variables[1]=VpostsComments;
        Variables[2]=Vshares;
        Variables[3]=VsharesComments;
        interval=Vinterval;

        addon.port.emit("vars", Variables,Vinterval);
        $("#messageboard").text( Messages.SettingsSaved);
    });
});
```

در همان ابتدای امر با استفاده از addon.port.on منتظر یک پیام رسان، به اسم vars می‌شود تا اطلاعات آن را دریافت کند که در اینجا بلافاصله بعد از نمایش پنل، اطلاعات برای آن ارسال شده و در صفحه، جایگذاری می‌کند. در قسمت رویداد کلیک دکمه ذخیره هم با استفاده از addon.port.emit اطلاعاتی را که کاربر به روز کرده است، به یک پیام رسان می‌دهیم تا برای آن سمت نیز ارسال کند تا در آن سمت، تنظیمات جدید ذخیره و جایگزین شوند.

نکته بسیار مهم: در کد بالا ما فایل جاوااسکریپت را از طریق فایل popup.html معرفی کردیم، نه از طریق خصوصیت contentscriptfile. این نکته را همیشه به خاطر داشته باشید. فایل‌های js خود را تنها در دو حالت استفاده کنید: از طریق دادن رشته به خصوصیت contentScript و استفاده از self به جای addon معرفی فایل js داخل خود فایل html با تگ script که به درد اسکریپت‌های با کد زیاد می‌خورد.

اگر فایل شما شامل استفاده از کلمه‌ی کلیدی addon نمی‌شود، می‌توانید فایل js خود را از طریق contentScriptFile هم اعمال کنید. فایل popup.html

```
<script src="jquery.min.js"></script> <!-- Including jQuery -->
<script type="text/javascript" src="const.js"></script>
<script type="text/javascript" src="popup.js"></script>
```

خواندن فید RSS سایت

خواندن فید سایت توسط فایل Rssreader.js انجام می‌شود که تمام اسکریپت‌های مورد نیاز برای اجرای آن، توسط

background.htm صدا زده شده است:

```
<script type="text/javascript" src="const.js"></script>
<script type="text/javascript" src="jquery.min.js"></script>
<script type="text/javascript" src="https://www.google.com/jsapi"></script>
<script type="text/javascript" src="rssreader.js"></script>
```

تنها کاری که باید انجام دهیم اجرای این فایل به عنوان یک فرآیند پس زمینه است. در کروم ما عادت داشتیم برای این کار در فایل manifest.json از خصوصیت background استفاده کنیم، ولی از آنجا که خود فایل main.js یک فایل اسکریپتی است که در پس زمینه اجرا می‌شود، طبق منابع موجود در نت چنین چیزی وجود ندارد و این فرآیند را به خود فایل main.js مربوط می‌دانستند. ولی من با استفاده از [page worker](#) چنین خصوصیتی را پیاده سازی کردم. page worker وظیفه دارد تا یک آدرس یا فایلی را در یک تب پنهان و در پشت صحنه اجرا کرده و به شما اجازه‌ی استفاده از DOM آن بدهد. نحوه‌ی دسترسی به فایل background.htm توسط page worker به صورت زیر تعریف می‌شود:

```
pageWorker = require("sdk/page-worker");
page= pageWorker.Page({
  contentScriptWhen: "ready",
  contentURL: self.data.url("./background.htm")
});
page.port.emit("vars",ss.storage.Variables,ss.storage.DateVariables,ss.storage.interval);
```

در فایل بالا شیء pageworker ساخته شد و درخواست یک پیچ نهان را برای فایل background.htm در دایرکتوری data می‌کند. استفاده از گزینه‌ی [contentScriptWhen](#) برای دسترسی به شیء addon در فایل‌های جاوااسکریپتی که استفاده می‌کنید ضروری است. در صورتی که حذف شود و نوشته نشود با خطای *addon is not defined* روبرو خواهید شد، چرا که هنوز این شیء شناسایی نشده است. در خط نهایی هم برای آن سمت یک پیام ارسال شده که حاوی مقادیر ذخیره شده می‌باشد.

فایل RSSReader.js

در اینجا هم مانند مطلبی که برای کروم گذاشتیم، خواندن فید، در یک دوره‌ی زمانی اتفاق می‌افتد. در کروم ما از chrome.alarm استفاده می‌کردیم، ولی در فایرفاکس از همان تایمرهای جاوااسکریپتی بهره می‌بریم. کد زیر را به فایلی به اسم rssreader.js اضافه می‌کنیم:

```
var variables=[];
var datevariables=[];
var period_time=60000;
var timer;
google.load("feeds", "1");

$(document).ready(function () {
  addon.port.on("vars", function(vars,datevars,interval) {
    if (vars)
    {
      Variables=vars;
    }
    if (datevars)
    {
      datevariables=datevars;
    }
    if(interval)
    period_time=interval*60000;
    alarmManager();
  });
});

function alarmManager()
{
  timer = setInterval(Run,period_time);
}

function Run() {
  if(Variables[0]){RssReader(Links.postUrl,0, Messages.PostsUpdated);}
  if(Variables[1]){RssReader(Links.posts_commentsUrl,1,Messages.CommentsUpdated);}
  if(Variables[2]){RssReader(Links.sharesUrl,2,Messages.SharesUpdated);}
  if(Variables[3]){RssReader(Links.shares_CommentsUrl,3,Messages.SharesCommentsUpdated);}
}

function RssReader(URL,index,Message) {
```

```

        var feed = new google.feeds.Feed(URL);
        feed.setResultFormat(google.feeds.Feed.XML_FORMAT);
        feed.load(function (result) {
if(result!=null)
{
var strRssUpdate = result.xmlDocument.firstChild.firstChild.childNodes[5].textContent;
var RssUpdate=new Date(strRssUpdate);
var lastupdate=new Date(datevariables[index]);
if(RssUpdate>lastupdate)
{
datevariables[index]=strRssUpdate;
addon.port.emit("notification",datevariables,Message);
}
}
});
}

```

در خطوط بالا متغیرها تعریف و توابع گوگل بارگزاری می‌شوند. سپس توسط `addon.port` یک شنونده ایجاد شده، تا بتواند مقادیر ذخیره شده را بازیابی کند. این مقادیر شامل موارد زیر است:

چه بخش‌هایی از سایت باید بررسی شوند.

آخرین تاریخ تغییر هر کدام که در زمان نصب افزونه، تاریخ نصب افزونه می‌شود و با اولین به روز رسانی، تاریخ جدیدی جای آن را می‌گیرد.

دوره‌ی سیکل زمانی یا همان `interval` بر اساس دقیقه

پس از اینکه شنونده مقادیر را دریافت کرد، تابع `alarmManager` اجرا شده و یک تایمر ایجاد می‌کند. بر خلاف کروم که برای این کار `api` تدارک دیده بود، اینجا شما باید از تایمرهای خود `جاوااسکریپت` مانند `SetTimeout` یا `SetInterval` استفاده کنید. موقع دریافت `interval` یا `period_time` ما آن را در 60000 ضرب کردیم تا دقیقه تبدیل به میلی ثانیه شود؛ چرا که تایمر، زمان را بر حسب میلی ثانیه دریافت می‌کند. وظیفه تایمر این هست که در هر دوره‌ی زمانی تابع `Run` را اجرا کند.

Run

این تابع بررسی می‌کند کاربر درخواست بررسی چه قسمت‌هایی از سایت را دارد و به ازای هر کدام، اطلاعات آن را از طریق پارامترها به تابع `rssreader` داده تا هر قسمت جداگانه بررسی شود. این اطلاعات به ترتیب: لینک فید مورد نظر، اندیس آخرین تاریخ به روزرسانی آن قسمت، پیامی که باید در وقت به روزرسانی به کار نمایش داده شود.

RSSReader

این تابع را قبلاً در [این مقاله](#) توضیح دادیم. تنها تغییری که کرده است، بدنه‌ی شرط بررسی تاریخ است که در صورت موفقیت، تاریخ جدید، جایگزین تاریخ قبلی شده و یک پیام به فایل `main.js` ارسال می‌کند تا از آن درخواست ذخیره‌ی تاریخی جدید و همچنین ایجاد یک `notification` برای آگاه‌سازی کاربر کند. پس باز به فایل `main.js` رفته و شنونده آن را تعریف می‌کنیم:

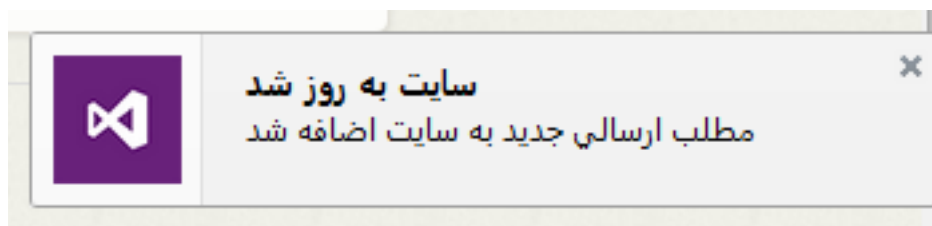
```

page.port.on("notification",function(lastupdate,Message)
{
ss.storage.DateVariables=lastupdate;
Make_a_Notification(Message);
})
function Make_a_Notification(Message)
{
var notifications = require("sdk/notifications");
notifications.notify({
title: "سایت به روز شد",
text: Message,
iconURL:self.data.url("./icon-64.png"),
data:"http://www.dotnettips.info",
onClick: function (data) {
tabs.open(data);
}
});
}

```

شنونده مورد نظر دو پارامتر تاریخ آخرین به روزرسانی را دریافت کرده و آن را جایگزین قبلی می‌کند و پیام را به تابع `Make_a_Notification` پاس می‌کند. پارامترهای ساخت نوتیفیکیشن به ترتیب شامل عنوان، متن پیام، آیکن و نهایتاً `data` است. دیتا شامل آدرس سایت است. زمانیکه کاربر روی نوتیفیکیشن کلیک می‌کند، استفاده شده و یک تب جدید را با آدرس سایت باز

می‌کنیم. به این ترتیب افزونه‌ی ما تکمیل می‌شود. برای اجرا و تست افزونه بر روی مرورگر فایرفاکس از دستور `cfx run` استفاده کنید.



البته این نکته قابل ذکر است که اگر کاربر اطلاعات پنل را به روزرسانی کند، تا وقتی که مرورگر بسته نشده و دوباره باز نشود تغییری نمی‌کند؛ چرا که ما تنها در ابتدای امر مقادیر ذخیره شده را به `RSSReader` فرستاده و اگر کاربر آن‌ها را به روز کند، ارسال پیام دیگری توسط `page worker` صورت نمی‌گیرد. پس کد موجود در `main.js` را به صورت زیر ویرایش می‌کنیم:

```
pageWorker = require("sdk/page-worker");
page= pageWorker.Page({
  contentScriptWhen: "ready",
  contentURL: self.data.url("./background.htm")
});

function SendData()
{
  page.port.emit("vars",ss.storage.Variables,ss.storage.DateVariables,ss.storage.interval);
}
SendData();
panel.port.on("vars", function (vars,interval) {
  ss.storage.Variables=vars;
  ss.storage.interval=interval;
  SendData();
});
```

در کد بالا ما خطی که به سمت `rssreader.js` پیام ارسال می‌کند را داخل یک تابع به اسم `SendData` قرار دادیم و بعد از تشکیل `page worker` آن را صدا زدیم و کد آن دقیقاً مانند قبل است؛ با این تفاوت که اینبار این تابع را در جای دیگری هم صدا می‌زنیم و آن زمانی است که برای پنل، پیام مقادیر جدید ارسال می‌شود که در آن پس از ذخیره موارد جدید تابع `SendData` را صدا می‌زنیم. پس موقع به روزرسانی هم مقادیر ارسال خواهند شد. مقادیر جدید به سمت `rssreader.js` رفته و تشکیل یک تایمر جدید را می‌دهند و البته چون قبلاً تایمر ایجاد شده است، پس باید چند خطی را هم به فایل `rssreader.js` اضافه کنیم تا تایمر قبلی را نابود کرده و تایمر جدیدی را ایجاد کند:

```
var timer;

function alarmManager()
{
  timer = setInterval(Run,period_time);
}

addon.port.on("vars", function(vars,datevars,interval) {
  if (vars)
  {
    Variables=vars;
  }
  if (datevars)
  {
    datevariables=datevars;
  }
  if(interval)
  period_time=interval*60000;

  if(timer!=null)
  {
    clearInterval(timer);
```



```

}
alarmManager();
});

```

در خط بالا متغیری به اسم timer ایجاد شده است که کد timer را در خود ذخیره می‌کند. پس موقع دریافت مقادیر بررسی میکنیم که اگر مقدار timer مخالف نال بود تایمر قبلی را با clearInterval از بین برده و تایمر جدیدی ایجاد کند. پس مشکل تایمری که از قبل موجود است نیز حل می‌گردد.

افزونه‌ی ما تکمیل شد. اجازه بدهید قبل از بستن بحث چندتا از موارد مهم موجود در sdk را نام ببریم:
Page Mod [page mod](#) موقعی که کاربر آدرسی را مطابق با الگویی (pattern) که ما دادیم، باز کند یک اسکریپت را اجرا خواهد کرد:

```

var pageMod = require("sdk/page-mod");

pageMod.PageMod({
  include: "*.mozilla.org",
  contentScript: 'window.alert("Page matches ruleset");'
});

```

```

var data = require("sdk/self").data;
var pageMod = require("sdk/page-mod");

pageMod.PageMod({
  include: "*.mozilla.org",
  contentScriptFile: [data.url("jquery-1.7.min.js"),
                     data.url("my-script.js")]
});

```

پنل تنظیمات

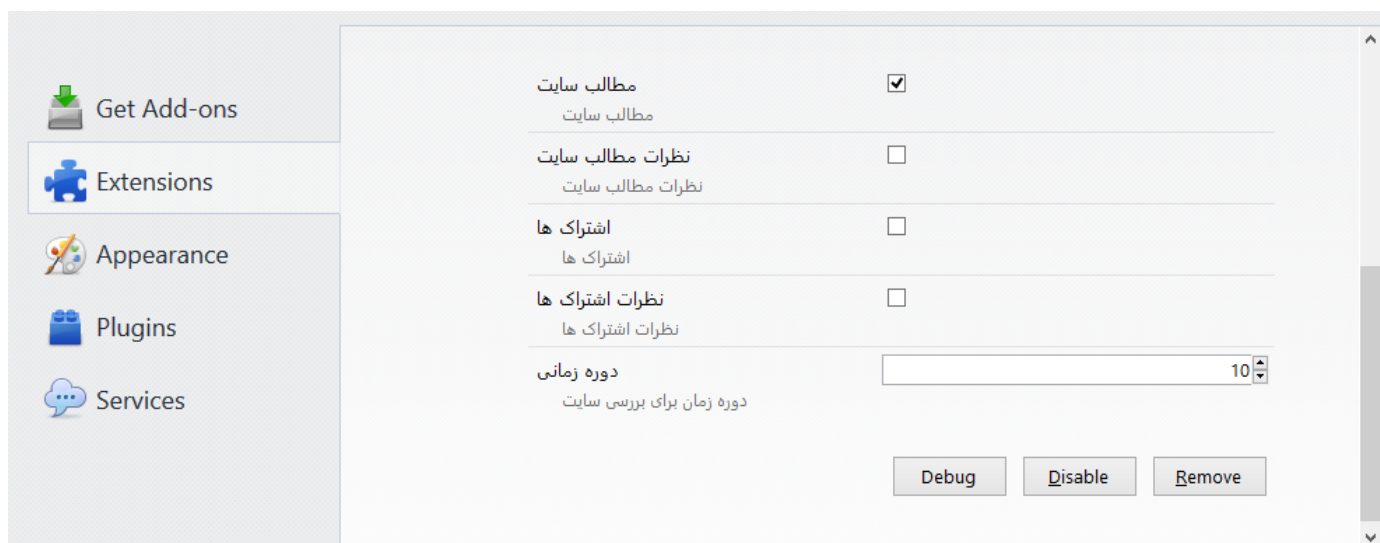
موقعی که شما افزونه‌ای را در فایرفاکس اضافه می‌کنید، در پنلی که مدیریت افزونه‌ها قرار دارد می‌توانید در تنظیمات هر افزونه، تغییری ایجاد کنید. برای ساخت چنین صفحه‌ای از خصوصیت [preferences](#) در فایل package.json کمک می‌گیریم که مقادیر به صورت آرایه ای داخل آن قرار می‌گیرند. مثال زیر پنج کنترل را به بخش تنظیمات افزونه اضافه می‌کند که چهار کنترل اول چک باکس Checkbox هستند؛ چرا که خصوصیت type آنها به bool ست شده است و شامل یک نام و عنوان یا برچسب label و یک توضیح کوتاه است و مقدار پیش فرض آن با خصوصیت value مشخص شده است. آخرین کنترل هم یک کادر عددی است؛ چرا که خاصیت type آن با integer مقداردهی شده و مقدار پیش فرض آن 10 می‌باشد.

```

"preferences": [{
  "description": "مطالب سایت",
  "type": "bool",
  "name": "post",
  "value": true,
  "title": "مطالب سایت"
},
{
  "description": "نظرات مطالب سایت",
  "type": "bool",
  "name": "postcomments",
  "value": false,
  "title": "نظرات مطالب سایت"
},
{
  "description": "اشتراک ها",
  "type": "bool",
  "name": "shares",
  "value": false,
  "title": "اشتراک ها"
},
{
  "description": "نظرات اشتراک ها",
  "type": "bool",
  "name": "sharescomments",
  "value": false,
  "title": "نظرات اشتراک ها"
},
{
  "description": "دوره زمان برای بررسی سایت",
  "name": "interval",
  "type": "integer",

```

```
"value": 10,
"title": "دوره زمانی"
}]
```



از آنجا که مقادیر بالا تنها مقادیر پیش فرض خودمان هست و اگر کاربر آن‌ها را تغییر دهد، در این صفحه هم باید اطلاعات تصحیح شوند، برای همین از کد زیر برای دسترسی به پنل تنظیمات و کنترل‌های موجود آن استفاده می‌کنیم. همانطور که می‌بینید کد مورد نظر را در یک تابع به نام Perf_Default_Value قرار دادیم و آن را در بدو اجرا صدا زدیم. پس کاربر اگر به پنل تنظیمات رجوع کند، می‌تواند تغییراتی را که قبلاً داده است، ببیند. بنابراین اگر الان تغییری را ایجاد کند، تا باز شدن مجدد مرورگر چیزی نمایش داده نمی‌شود. برای همین دقیقاً مانند تابع SendData این تابع را هم در کد شوند پنل panel اضافه می‌کنیم؛ تا اگر کاربر اطلاعات را از طریق روش قبلی تغییر داد، اطلاعات هم اینک به روز شوند.

```
function Perf_Default_Value()
{
var preferences = require("sdk/simple-prefs").prefs;

preferences.post = ss.storage.Variables[0];
preferences.postcomments = ss.storage.Variables[1];
preferences.shares = ss.storage.Variables[2];
preferences.sharescomments = ss.storage.Variables[3];
preferences["myinterval"] = parseInt(ss.storage.interval);
}
Perf_Default_Value();

panel.port.on("vars", function (vars,interval) {
ss.storage.Variables=vars;
ss.storage.interval=interval;
SendData();
Perf_Default_Value();
});
```

البته کاربر فقط برای دیدن اطلاعات بالا به این صفحه‌ی تنظیمات نمی‌آید؛ بلکه بیشتر برای تغییر آن‌ها می‌آید. پس باید به تغییر مقدار کنترل‌ها گوش فرا دهیم. برای گوش دادن به تغییر تنظیمات، برای موقعی که کاربر قسمتی از تنظیمات را ذخیره کرد، از کدهای زیر بهره می‌بریم:

```
perf=require("sdk/simple-prefs");
var preferences = perf.prefs;
function onPrefChange(prefName) {

switch(prefName)
{
```

```
case "post":
ss.storage.Variables[0]=preferences[prefName];
break;
case "postcomments":
ss.storage.Variables[1]=preferences[prefName];
break;
case "shares":
ss.storage.Variables[2]=preferences[prefName];
break;
case "sharescomments":
ss.storage.Variables[3]=preferences[prefName];
break;
case "myinterval":
ss.storage.interval=preferences[prefName];
break;
}
}
//perf.on("post", onPrefChange);
//perf.on("postcomments", onPrefChange);
perf.on("", onPrefChange);
```

متد on دو پارامتر دارد: اولی، نام کنترل مورد نظر که با خصوصیت name تعریف کردیم و دومی هم تابع callback آن می‌باشد و در صورتی که پارامتر اول با "" مقداردهی شود، هر تغییری که در هر کنترلی رخ بدهد، تابع callback صدا زده می‌شود. از آنجا که نام کنترل‌ها به صورت string برگشت داده می‌شوند، برای دسترسی به مقادیر موجود در تنظیمات از همان روش داخل [] بهره می‌گیریم. مقادیر را گرفته و داخل storage ذخیره می‌کنیم.

اشکال زدایی Debug

یکی از روش‌های اشکال زدایی، استفاده از console.log هست که میتونید برای بازبینی مقادیر و وضعیت‌ها، از آن استفاده کنید که نتیجه‌ی آن داخل کنسول نمایش داده می‌شود و اگر هم در برنامه خطایی رخ دهد، داخل کنسول به شما نمایش خواهد داد.

[سورس کار](#)

نظرات خوانندگان

نویسنده: بهمن خلفی
تاریخ: ۱۳۹۳/۱۱/۱۵ ۱۲:۳

از مطالب جذاب و کامل شما بسیار سپاسگذارم. چند نکته هست اگر امکان دارد آنها را نیز پوشش دهید
مثلا ارتباط این افزونه‌ها با بانکهای اطلاعاتی (مانند : localStorage مرورگر یا منابع داده دیگر مثل MySQL یا SQL Server و ...) و نحوه ذخیره سازی داده ها.
مجددا متشکرم.

نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۱۱/۱۵ ۱۵:۴

در مورد ذخیره سازی لوکال مرورگر که در بالا همان اول مقاله توضیح دادم و در کروم هم که گفتیم با کد زیر اینکارو انجام میدیم:

```
chrome.storage.local.set  
chrome.storage.sync.set
```

این نکته را هم خاطرنشان کنم که در فایرفاکس [ذخیره مقادیر](#) تا حجم حدودی 5 مگابایت میسر است در مورد اتصال به دیتابیس sqlite میتونید از این [لینک](#) کمک بگیرید که به موارد دیگه هم لینک شده و اگر دقت کنید می‌بینید که میتونید از کدهای ++c هم استفاده کنید و همینطور [اینجا](#) هم که یک نفر پرسش کرده و یکی هم پاسخش را داده.
در مورد بقیه اتصالات به بانک هایی چون sql server و ... هم میتوانید از طریق apiها یا وب سرویس‌ها عمل کنید که نیاز به یک فایل jquery برای اتصال به آنها دارید یا فریمورک‌های جاوااسکریپتی که در این زمینه مهیا شده است.
این [مقاله](#) هم ممکنه براتون جالب باشه

نویسنده: پیشگوی
تاریخ: ۱۳۹۴/۰۶/۰۱ ۱۵:۶

سلام مقاله جالبی بود. ممنون. برای ایجاد منوها برای کلیدهایی که ایجاد میکنیم چطوری میشه عمل کرد؟ مثلا فایرباگ و یا ColorZilla کلیدهایی هستن که یک آیکن کنارشون دارن و زیر منو تعریف شده براشون. ممنون میشم این مورد با ایجاد پنجره‌های بازشو آموزش بدید. البته منظورم پنجره‌های ویندوزی برای داخل فایرفاکس هستش که باز میشن نه پاپ آپی که تو آموزشهای اکستنشن برای فایرفاکس آموزش داده شده.

ممنون

نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۶/۰۱ ۱۶:۴۳

[این سوری](#) را ببینید
اگر منظورتان هم [این مطلب](#) بوده است.

نویسنده: پیشگوی
تاریخ: ۱۳۹۴/۰۶/۰۱ ۱۸:۱۷

بلی همون بوده. من در استفادش به مشکل خوردم. این خطا رو بهم میده:

```
My simpleAddon could not be verified for use in firefox. proceed with caution
```

از همون مثالش جلو رفتم ولی به جای b- من حالت عادیش رو جلو رفتم. ممنون میشم راهنمایی کنید

نویسنده: علی یگانه مقدم
تاریخ: ۱۹:۴۹ ۱۳۹۴/۰۶/۰۱

مشکل اصلی از کد شما نیست
مشکل اصلی این است که از فایرفاکس 40 به بعد به دلیلی سواستفاده هایی که از دزدیدن اطلاعات و نصب ابزارهای ناخواسته
پیش آمده است فایرفاکس نیاز دارد اکستنشن امضا شود
[اطلاعات بیشتر](#)

[مشکل مشابه](#)

نویسنده: علی یگانه مقدم
تاریخ: ۰:۲۷ ۱۳۹۴/۰۶/۰۳

[اینجا](#) و [اینجا](#) را ببینید