**Develop a Chrome Extension in 2018**

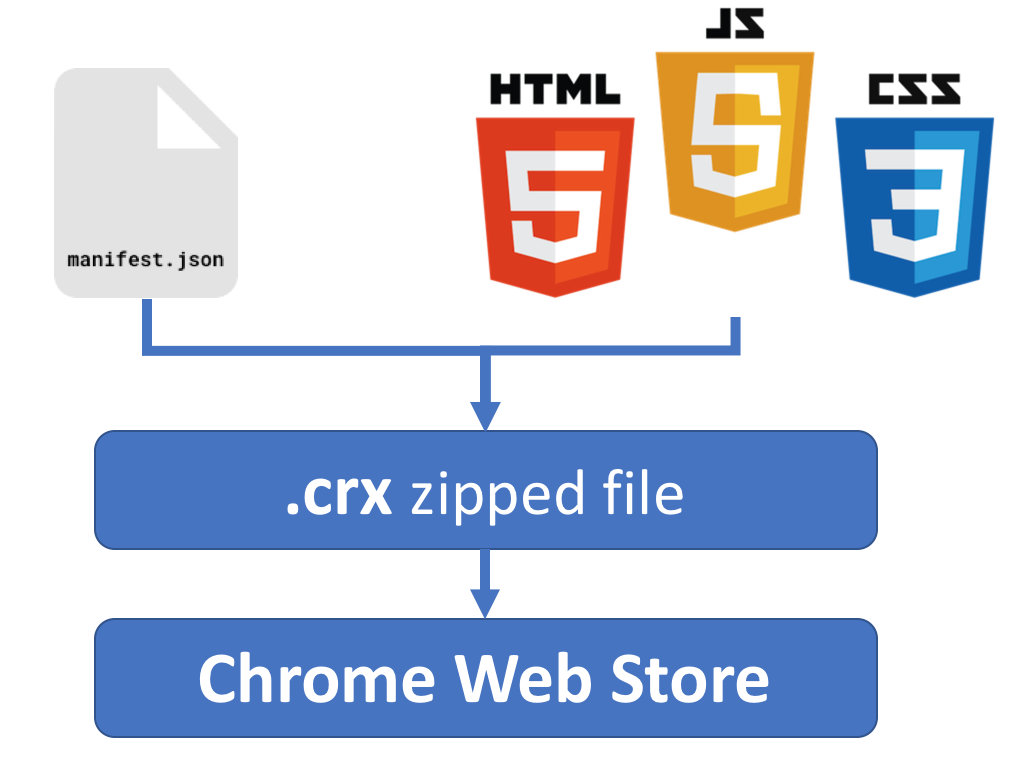
* **Why build a Chrome extension?**
* **Structure and Architecture**
* **Test and deploy your Chrome extension**
* **Chrome extension boilerplate by React**
* **Examples of Chrome extensions**

**Chrome Extension?**

Extensions are small software programs that customize the browsing experience. They enable users to tailor Chrome functionality and behavior to individual needs or preferences. They are built on web technologies such as HTML, JavaScript, and CSS.

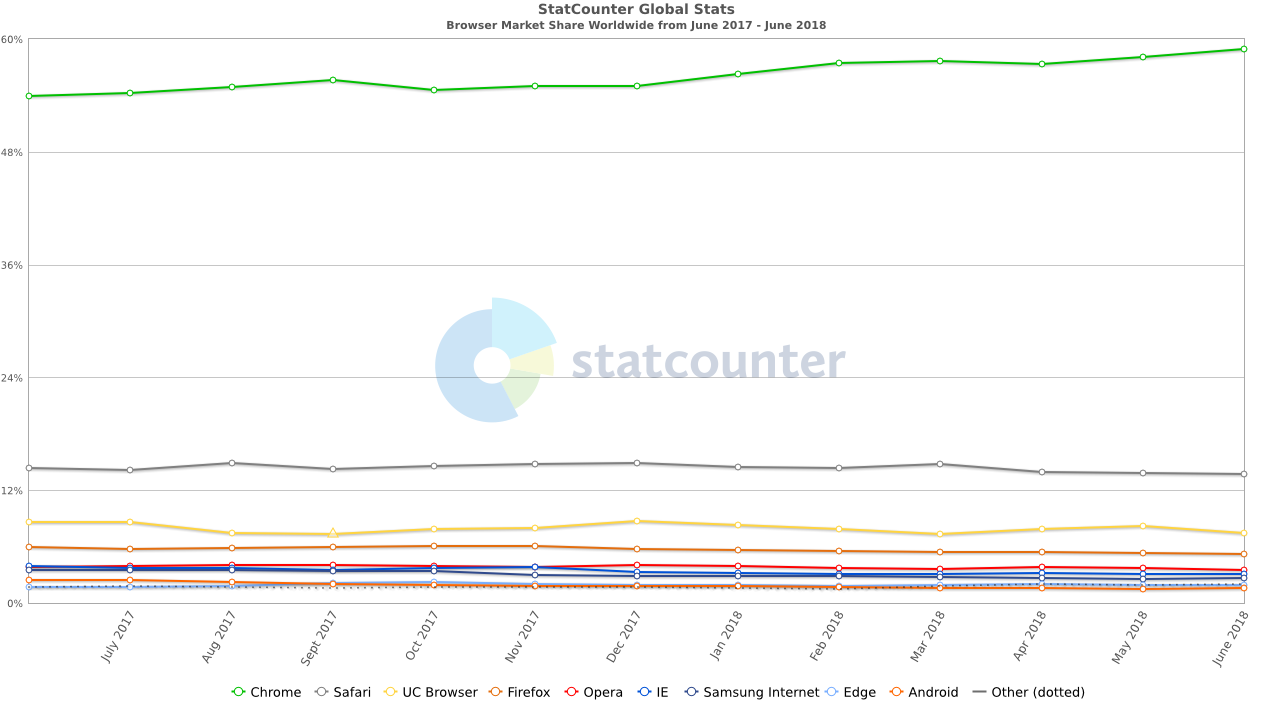
Extension files are zipped into a single .crx package that the user downloads and installs. This means extensions do not depend on content from the web, unlike ordinary web apps.

Extensions are distributed through the [**Chrome Developer Dashboard**](https://chrome.google.com/webstore/developer/dashboard) and published to the [**Chrome Web Store**](http://chrome.google.com/webstore). For more information, see the [**store developer documentation**](http://code.google.com/chrome/webstore).



**Why build a Chrome extension?**

* Chrome – most popular browser
* Integration with Chrome
* Chrome APIs
* Could Extend Chrome & any web site
* Standard Web Stack (JS, HTML, CSS)



**Manifest file**

Every single Chrome extension must have a JSON-formatted manifest file, named ***manifest.json***. The manifest file, gives the browser information about the extension, such as the most important files and the capabilities the extension might use.  
This manifest file provides all necessary information about your Chrome extension, such as versioning, permissions, and other useful meta information. The file must be placed in the root folder of your chrome extension.

According to Google, every manifest file must contain the following information:

{

"manifest\_version": "2.1",

"name": "My Extension",

"version": "versionString"

}

The following information is optional but recommended:

{

"description": "Gets information from Google.",

"icons": {

"128": "icon\_128.png"

},

"background": {

"persistent": false,

"scripts": ["bg.js"]

},

"permissions": ["http://\*.google.com/", "https://\*.google.com/"],

"browser\_action": {

"default\_title": "",

"default\_icon": "icon\_19.png",

"default\_popup": "popup.html"

}

}

For further information on how to create a manifest file, I recommend checking out the documentation from Google [here](https://developer.chrome.com/extensions/manifest).

**Architecture**

An extension’s architecture will depend on its functionality, but many robust extensions will include multiple components:

* Manifest ([read more](https://developer.chrome.com/extensions/manifest))
* Background Script ([read more](https://developer.chrome.com/extensions/overview#background_script))
* UI Elements ([read more](https://developer.chrome.com/extensions/overview#pages))
* Content Script ([read more](https://developer.chrome.com/extensions/overview#contentScripts))
* Options Page ([read more](https://developer.chrome.com/extensions/overview#optionsPage))

We learned Manifest before. Let’s take a look on other stuff.

**Background Script**

The [**background script**](https://developer.chrome.com/background_pages) is the extension's event handler; it contains listeners for browser events that are important to the extension. It lies dormant until an event is fired then performs the instructed logic. An effective background script is only loaded when it is needed and unloaded when it goes idle.

{

"name": "Awesome Test Extension",

...

"background": {

"scripts": ["background.js"],

"persistent": false

},

...

}

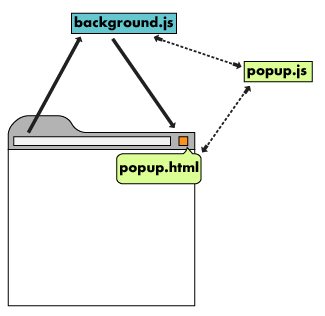
### UI Elements

An [**extension's user interface**](https://developer.chrome.com/extensions/user_interface) should be purposeful and minimal. The UI should customize or enhance the browsing experience without distracting from it.

Most extensions have a [**browser action**](https://developer.chrome.com/extensions/browserAction) or [**page action**](https://developer.chrome.com/extensions/pageAction), but can contain other forms of UI, such as [**context menus**](https://developer.chrome.com/contextMenus), use of the [**omnibox**](https://developer.chrome.com/extensions/omniBox), or creation of a [**keyboard shortcut**](https://developer.chrome.com/extensions/commands).

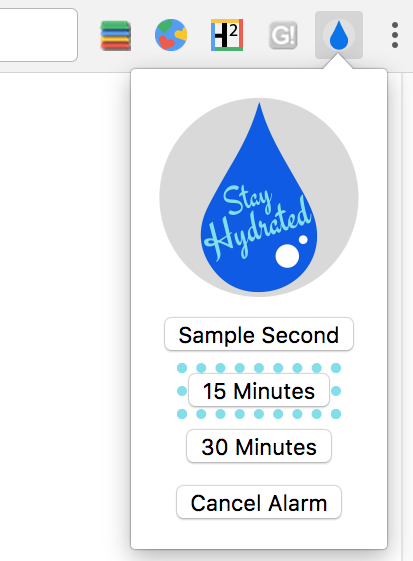
Extension UI pages, such as a [**popup**](https://developer.chrome.com/extensions/user_interface#popup), can contain ordinary HTML pages with JavaScript logic. Extensions can also call **[tabs.create](https://developer.chrome.com/extensions/tabs" \l "method-create)** or window.open() to display additional HTML files present in the extension.

* Popup
* Tooltip
* Context Menu
* Omnibox



### Popup

A popup is an HTML file that is displayed in a special window when the user clicks the toolbar icon. A popup works very similarly to a web page; it can contain links to stylesheets and script tags, but does not allow inline JavaScript.



The popup can be registered in the manifest, under browser action or page action.

{

"name": "Drink Water Event",

...

"browser\_action": {

"default\_popup": "popup.html"

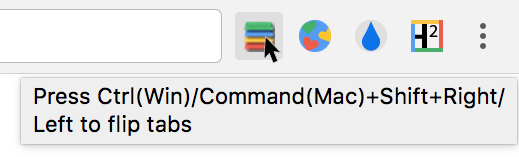
}

...

}

### Tooltip

Use a tooltip to give short descriptions or instructions to users when hovering over the browser icon.



Tooltips are registered in the "default\_title" field **[browser\_action](https://developer.chrome.com/browserAction)** or **[page\_action](https://developer.chrome.com/pageAction)** in the manifest.

{

"name": "Tab Flipper",

...

"browser\_action": {

"default\_title": "Press Ctrl(Win)/Command(Mac)+Shift+Right/Left to flip tabs"

}

...

}

Specialized locale strings are implemented with [**Internationalization**](https://developer.chrome.com/i18n). Create directories to house language specific messages within a folder called \_locales. The following image shows a file path for an extension that supports English and Spanish locales.



[**Format messages**](https://developer.chrome.com/i18n-messages) inside of each language's messages.json.

{

"\_\_MSG\_tooltip\_\_": {

"message": "Hola!",

"description": "Tooltip Greeting."

}

}

### Context Menu

Add new [**context menu**](https://developer.chrome.com/contextMenus) options by granting the "contextMenus" permission in the manifest.

{

"name": "Global Google Search",

...

"permissions": ["contextMenus", "storage"],

"icons": {

"16": "globalGoogle16.png",

"48": "globalGoogle48.png",

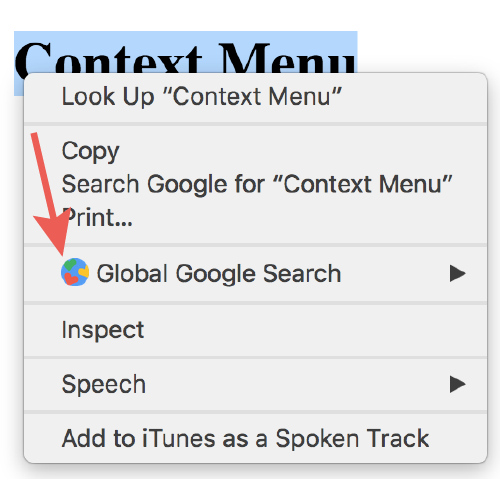
"128": "globalGoogle128.png"

}

...

}

The 16x16 icon is displayed next to the new menu entry.



Create a context menu by calling **[contextMenus.create](https://developer.chrome.com/contextMenus" \l "method-create)** in the [**background script**](https://developer.chrome.com/background_pages). This should be done under the **[runtime.onInstalled](https://developer.chrome.com/runtime" \l "event-onInstalled)** listener event.

chrome.runtime.onInstalled.addListener(function () {

for (let key of Object.keys(kLocales)) {

chrome.contextMenus.create({

id: key,

title: kLocales[key],

type: 'normal',

contexts: ['selection'],

});

}

});

const kLocales = {

'com.au': 'Australia',

'com.br': 'Brazil',

'ca': 'Canada',

'cn': 'China',

'fr': 'France',

'it': 'Italy',

'co.in': 'India',

'co.jp': 'Japan',

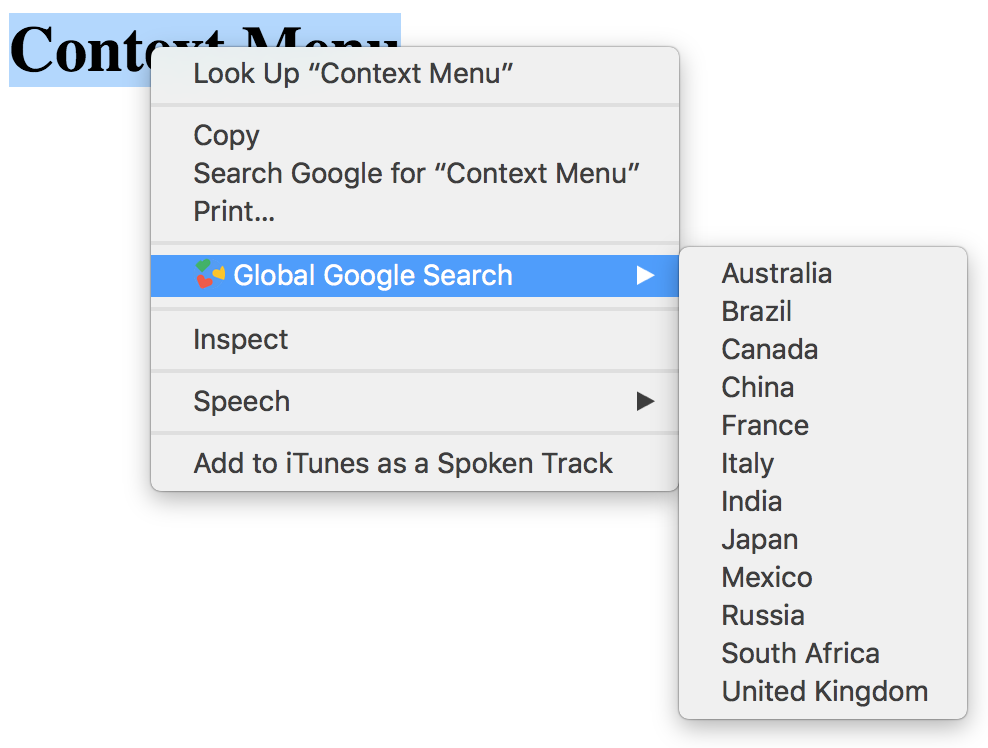
'com.ms': 'Mexico',

'ru': 'Russia',

'co.za': 'South Africa',

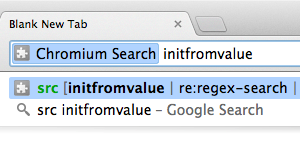
'co.uk': 'United Kingdom'

};



### Omnibox

### The omnibox API allows you to register a keyword with Google Chrome's address bar, which is also known as the omnibox. When the user enters your extension's keyword, the user starts interacting solely with your extension. Each keystroke is sent to your extension, and you can provide suggestions in response. ([Read more](https://developer.chrome.com/extensions/omnibox)).

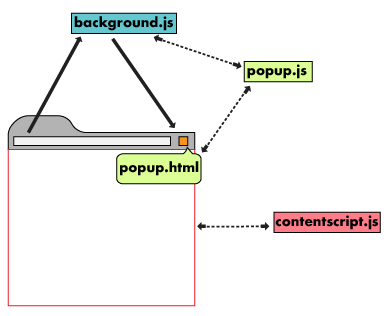


### Content scripts

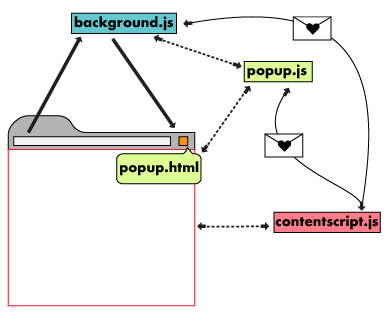
Extensions that read or write to web pages utilize a [**content script**](https://developer.chrome.com/extensions/content_scripts). The content script contains JavaScript that executes in the contexts of a page that has been loaded into the browser. Content scripts read and modify the DOM of web pages the browser visits.

A content script has access to the current page, but is limited in the APIs it can access. For example, it cannot listen for clicks on the browser action. We need to add a different type of script to our extension, a background script, which has access to every Chrome API but cannot access the current page.

Content scripts have some limitations. They cannot use ***chrome.\* APIs***, with the exception of ***extension***, ***i18n***, ***runtime***, and ***storage***.



Content scripts can communicate with their parent extension by exchanging [**messages**](https://developer.chrome.com/extensions/messaging) and storing values using the [**storage**](https://developer.chrome.com/extensions/storage) API.



### Options Page

Just as extensions allow users to customize the Chrome browser, the [**options page**](https://developer.chrome.com/extensions/options) enables customization of the extension. Options can be used to enable features and allow users to choose what functionality is relevant to their needs.

{

"name": "My Extension Name",

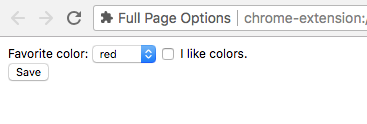
...

"options\_page": "options.html",

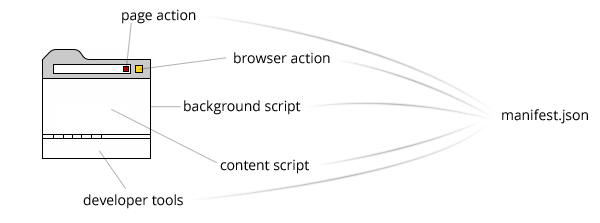
...

}

Allow users to customise the behavior of an extension by providing an options page. A user can view an extension's options by right-clicking the extension icon in the toolbar then selecting options or by navigating to the extension management page at chrome://extensions, locating the desired extension, clicking **Details**, then selection the options link.



Here's a diagram of the architecture for a Chrome extension:



The ***page\_action*** property is similar to the browser action, but the icon is shown inside the address bar:



## **Chrome APIs**

In addition to having access to the same APIs as web pages, extensions can also use [**extension-specific APIs**](https://developer.chrome.com/extensions/api_index) that create tight integration with the browser. Extensions and webpages can both access the standardwindow.open() method to open a URL, but extensions can specify which window that URL should be displayed in by using the Chrome API **[tabs.create](https://developer.chrome.com/extensions/tabs" \l "method-create)** method instead.

For more information, explore the [**Chrome API reference docs**](https://developer.chrome.com/extensions/api_index)

Chrome provides extensions with many special-purpose APIs like chrome.runtime and chrome.alarms.

* Stable APIs
* Beta APIs
* Dev APIs
* Experimental APIs

## **Stable APIs**

* [**alarms**](https://developer.chrome.com/extensions/alarms)

Use the chrome.alarms API to schedule code to run periodically or at a specified time in the future.

* [**bookmarks**](https://developer.chrome.com/extensions/bookmarks)

Use the chrome.bookmarks API to create, organize, and otherwise manipulate bookmarks. Also see [**Override Pages**](https://developer.chrome.com/extensions/override), which you can use to create a custom Bookmark Manager page.

* [**browserAction**](https://developer.chrome.com/extensions/browserAction)

Use browser actions to put icons in the main Google Chrome toolbar, to the right of the address bar. In addition to its [**icon**](https://developer.chrome.com/extensions/browserAction#icon), a browser action can also have a [**tooltip**](https://developer.chrome.com/extensions/browserAction#tooltip), a [**badge**](https://developer.chrome.com/extensions/browserAction#badge), and a [**popup**](https://developer.chrome.com/extensions/browserAction#popups).

* [**browsingData**](https://developer.chrome.com/extensions/browsingData)

Use the chrome.browsingData API to remove browsing data from a user's local profile.

* [**commands**](https://developer.chrome.com/extensions/commands)

Use the commands API to add keyboard shortcuts that trigger actions in your extension, for example, an action to open the browser action or send a command to the extension.

* [**contentSettings**](https://developer.chrome.com/extensions/contentSettings)

Use the chrome.contentSettings API to change settings that control whether websites can use features such as cookies, JavaScript, and plugins. More generally speaking, content settings allow you to customize Chrome's behavior on a per-site basis instead of globally.

* [**contextMenus**](https://developer.chrome.com/extensions/contextMenus)

Use the chrome.contextMenus API to add items to Google Chrome's context menu. You can choose what types of objects your context menu additions apply to, such as images, hyperlinks, and pages.

* [**cookies**](https://developer.chrome.com/extensions/cookies)

Use the chrome.cookies API to query and modify cookies, and to be notified when they change.

* [**debugger**](https://developer.chrome.com/extensions/debugger)

The chrome.debugger API serves as an alternate transport for Chrome's [**remote debugging protocol**](https://developer.chrome.com/devtools/docs/debugger-protocol). Use chrome.debugger to attach to one or more tabs to instrument network interaction, debug JavaScript, mutate the DOM and CSS, etc. Use the Debuggee tabId to target tabs with sendCommand and route events by tabId from onEvent callbacks.

* [**declarativeContent**](https://developer.chrome.com/extensions/declarativeContent)

Use the chrome.declarativeContent API to take actions depending on the content of a page, without requiring permission to read the page's content.

* [**desktopCapture**](https://developer.chrome.com/extensions/desktopCapture)

Desktop Capture API that can be used to capture content of screen, individual windows or tabs.

* [**documentScan**](https://developer.chrome.com/extensions/documentScan)

Use the chrome.documentScan API to discover and retrieve images from attached paper document scanners.

* [**downloads**](https://developer.chrome.com/extensions/downloads)

Use the chrome.downloads API to programmatically initiate, monitor, manipulate, and search for downloads.

* [**events**](https://developer.chrome.com/extensions/events)

The chrome.events namespace contains common types used by APIs dispatching events to notify you when something interesting happens.

* [**extension**](https://developer.chrome.com/extensions/extension)

The chrome.extension API has utilities that can be used by any extension page. It includes support for exchanging messages between an extension and its content scripts or between extensions, as described in detail in [**Message Passing**](https://developer.chrome.com/extensions/messaging).

* [**history**](https://developer.chrome.com/extensions/history)

Use the chrome.history API to interact with the browser's record of visited pages. You can add, remove, and query for URLs in the browser's history. To override the history page with your own version, see [**Override Pages**](https://developer.chrome.com/extensions/override).

* [**i18n**](https://developer.chrome.com/extensions/i18n)

Use the chrome.i18n infrastructure to implement internationalization across your whole app or extension.

* [**identity**](https://developer.chrome.com/extensions/identity)

Use the chrome.identity API to get OAuth2 access tokens.

* [**notifications**](https://developer.chrome.com/extensions/notifications)

Use the chrome.notifications API to create rich notifications using templates and show these notifications to users in the system tray.

* [**pageAction**](https://developer.chrome.com/extensions/pageAction)

Use the chrome.pageAction API to put icons in the main Google Chrome toolbar, to the right of the address bar. Page actions represent actions that can be taken on the current page, but that aren't applicable to all pages. Page actions appear grayed out when inactive.

* [**permissions**](https://developer.chrome.com/extensions/permissions)

Use the chrome.permissions API to request [**declared optional permissions**](https://developer.chrome.com/extensions/permissions#manifest) at run time rather than install time, so users understand why the permissions are needed and grant only those that are necessary.

* [**power**](https://developer.chrome.com/extensions/power)

Use the chrome.power API to override the system's power management features.

* [**printerProvider**](https://developer.chrome.com/extensions/printerProvider)

The chrome.printerProvider API exposes events used by print manager to query printers controlled by extensions, to query their capabilities and to submit print jobs to these printers.

* [**sessions**](https://developer.chrome.com/extensions/sessions)

Use the chrome.sessions API to query and restore tabs and windows from a browsing session.

* [**storage**](https://developer.chrome.com/extensions/storage)

Use the chrome.storage API to store, retrieve, and track changes to user data.

* [**tabs**](https://developer.chrome.com/extensions/tabs)

Use the chrome.tabs API to interact with the browser's tab system. You can use this API to create, modify, and rearrange tabs in the browser.

* [**vpnProvider**](https://developer.chrome.com/extensions/vpnProvider)

Use the chrome.vpnProvider API to implement a VPN client.

* [**webRequest**](https://developer.chrome.com/extensions/webRequest)

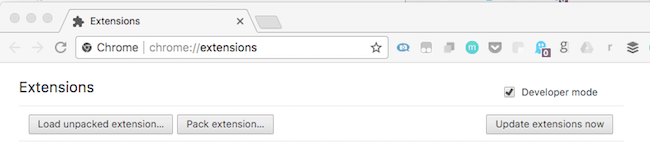
Use the chrome.webRequest API to observe and analyze traffic and to intercept, block, or modify requests in-flight.

* [**windows**](https://developer.chrome.com/extensions/windows)

Use the chrome.windows API to interact with browser windows. You can use this API to create, modify, and rearrange windows in the browser.

**Test Chrome extension**

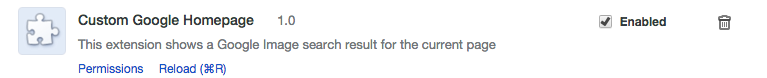
Testing your new extension is pretty straight-forward. Once you’ve activated the “developer mode” of your chrome extension settings (accessible with *chrome://extensions*).



You can simply add your unpacked extension to your Chrome browser to test it. If successfully added, you should see the icon of your own extension within your browser.

To test your newly created extension you can perform any functionality and acceptance test to see if it works as intended.

1. Go to ***chrome://extensions*** in your browser
2. Ensure that the Developer mode checkbox in the top right-hand corner is checked.
3. Click Load unpacked extension to pop up a file-selection dialog & select your directory.
4. Ensure that the enabled box next to your chrome extension is checked so you can see it in action.



**Chrome Extension Simple Examples** - <https://developer.chrome.com/extensions/samples>

**Chrome Extension React Example** –

<https://github.com/FullStack-Academy-Kiev/react-chrome-extension>