[**How to Build a Chrome Extension**](http://lifehacker.com/5857721/how-to-build-a-chrome-extension)

Google Chrome is the best web browser around right now, and part of Chrome's appeal is owed to（归功于） its excellent extensions. The good news: It's not that hard to get started making your own Chrome extensions. In this guide, we'll take you from the most simple Hello World extension (no HTML or JavaScript knowledge required) to a more complex RSS-fetching extension to get you started down your path as a Chrome-extension-making guru（专家）.

I released my first Chrome extension—a Google Music power-up extension called [Music Plus](https://chrome.google.com/webstore/detail/ipfnecmlncaiipncipkgijboddcdmego)—this summer, followed by a simpler [Lifehacker Notifier extension](https://chrome.google.com/webstore/detail/hjfpocpmopdhibcplfhjpceegcchfokp) that monitors Lifehacker's RSS feed, displays notifications of new posts in HTML 5 popups or badges, and so on.

Today, we'll walk through how to make your first and simplest Hello World Chrome extension, then we'll make a light version of the Lifehacker Notifier extension that fetches an RSS feed and displays feed items in a popup window when you click your extension's button. So let's get extending!

***Note:****While the Hello World extension requires zero knowledge of JavaScript or HTML (and, frankly, you could have a lot of fun with just that if you wanted), this second half of this post requires an understanding of both. To make it to the end of the post, some experience will be necessary. We've also previously shown you*[*how to build a Firefox extension*](http://lifehacker.com/264490/how-to-build-a-firefox-extension)*, so if Firefox is more your speed, you may want to check out that guide.*

**What You Need to Know Before You Start**

If you're comfortable putting together a web site—that is, you know a little HTML and are familiar with JavaScript—you can make a Chrome extension. If you're new to HTML and JavaScript, our beginner's guides for [learning to code](http://lifehacker.com/5744113/learn-to-code-the-full-beginners-guide) and [how to make a web site](http://lifehacker.com/5790955/how-to-make-a-web-site-the-complete-guide) are great starting points.

Beyond those two core competencies（技能）, there's really nothing special you need to know. Chrome extensions are delightfully easy to make if you've ever spent any time making web pages or hacking away with JavaScript, and even if you're only a beginner in those arenas, you can probably pull off a Chrome extension. So let's do just that.

**Our Project: From "Hello World" to RSS Fetcher**

For the purpose of this guide, we're going to start with a "Hello World" extension. If you're new to programming, the classic "Hello World" program is a rite of passage for getting started with any language, framework, or project, and its goal is simply to create something capable of outputting the text "Hello World".

After we finish our "Hello World" project, I'll walk you through the basics of making an RSS fetching extension along the lines of the [Lifehacker Notifier extension](https://chrome.google.com/webstore/detail/hjfpocpmopdhibcplfhjpceegcchfokp) for Chrome. Basically this extension adds a button to your toolbar, monitors Lifehacker's RSS feed, and displays our posts in a handy drop-down. It also shows a popup when a new post appears and displays an unread badge on the extension button. So let's get started.

**Manifest（清单）.json: The Cornerstone（基石） of Your Chrome Extension**

Every Chrome extension consists of, at minimum, a file calledmanifest.json, which defines the basics of your extension—its name, description, version number, what kind of extension it is (there are a handful of different things a Chrome extension can do, which we'll talk more about below), the permissions it needs to run (e.g., what web sites it needs access to), and so on.

So let's get started. Create a new folder—let's name itHello World—and, using your favorite text editor, create a new text file calledmanifest.json. Copy and paste the following code into it:

{

"name": "Hello World!",

"version": "1.0",

"description": "My first Chrome extension.",

"browser\_action": {

"default\_icon": "icon.png"

}

}

The name, version, and description are all pretty self-explanatory, butbrowser\_actionis something new! Thebrowser\_actionproperty tells Chrome that we're making an extension that will add a button to the toolbar. So far, all we've done is assign an icon to that button. Basically you're telling Chrome that your extension has an icon, calledicon.png, and that it's located in the same folder as yourmanifest.jsonfile. Of course, you don't have anicon.pngfile in yourHello Worldfolder just yet, so let's fix that.

Download [this tiny image](http://cache.gawker.com/assets/images/lifehacker/2011/11/icon.png) ([via](http://www.psdgraphics.com/icons/photoshop-world-globe-icon/)) and copy it to your Hello World folder.

We haven't done much yet, but you've actually already made something you can test out, so let's do that. Point Chrome to <chrome://extensions/>, tick the Developer mode checkbox at the top-left of that window, then click the Load unpacked extension button. Point Chrome to your Hello World folder, click Select or OK (varies by operating system), Chrome will load up your stub of an extension, and you should see your little globe icon appear in your Chrome toolbar thusly:

If you try clicking your Hello World button, however, you'll notice it does an unsurprising amount of nothing. Let's fix that.

**Creating a Browser Action, or, Wouldn't It Be Nice if That Button Did Something?**

Open up yourmanifest.jsonfile and add apopupaction that points to an HTML file calledpopup.html, like so:

{

"name": "Hello World!",

"version": "1.0",

"description": "My first Chrome extension.",

"browser\_action": {

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

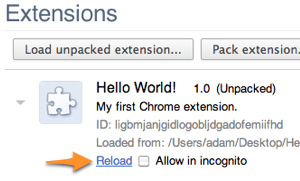
"popup": "popup.html"

}

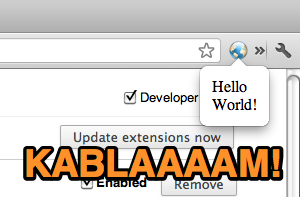
}

Note: The filenames don't matter as long as you're pointing Chrome to the right files—you could call ithelloworld.htmlinmanifest.jsonas long as you also named your filehelloworld.html.

Speaking of, you now need to createpopup.html. So, once again, create an HTML file calledpopup.htmland save it in yourHello Worldfolder. Insidepopup.html, simply add the text "Hello World!" (In theory you'd want to put some valid HTML markup in there, but it's not strictly necessary, so we're going to skip it here.)

Make sure you've savedmanifest.jsonandpopup.html, head back to <chrome://extensions/>, click the expand icon next to the Hello World extension in the Extension list, then click the Reload link (pictured at right).

Chrome will reload the extension using your updated code. Now click the button and get ready for the money shot!



Achievement unlocked! You've Hello World-ed your first Chrome extension. Nice work. Now let's take things up a notch. (If you had any trouble along the way, download my working [Hello World.zip](http://adampash.com/s/Hello%20World.zip) and compare it with yours.)

**Where Do We Go From Here?**

So Chrome can add buttons to your toolbar, you can click those buttons, and something can happen. Neato! You could stop your extension development here and have plenty of fun installing extensions on your friends' computers—add a little HTML image markup in place of the "Hello World!" text and you've unlocked a prankster's delight! Imagine the possibilities. Your unsuspecting friend clicks an alluring new button and—slam, goatse! Or toss in a YouTube video and... RICKROLL! (Or don't do this. Your friends won't thank you, but they might if they knew what you weren't subjecting them to.)

Impressing your friends with your sparkling sense of humor will only take you so far, though. You've probably seen the basic click-button-show-drop-down behavior in tons of different Chrome extensions before. But you've also seen extensions that perform very different tasks. If you take a gander（一瞥） at the [Chrome Extension Developer's Guide](http://code.google.com/chrome/extensions/devguide.html), you'll see the familiar Browser Actions at the top of the list, but you'll also notice a ton of other things your extension can do, from creating desktop notifications or adding a keyword to the omnibox（地址栏） to creating an options page or performing actions that modify specific web pages. When you're ready to dive deep into extension development, you'll want to page through the developer documentation to get a feel for what your extension might take advantage of. You may also want to take a look at the[manifest.json documentation](http://code.google.com/chrome/extensions/manifest.html) to get a feel for some of your other options available to yourmanifest.jsonfile.

For now, we're going to dive a little deeper into the browser actions. It's time to make yourself an RSS-reading and -notifying Chrome extension. So let's do it.

**It's Time to Next Level This Hog**

Now that you've safely navigated from zero to Hello World, we're going to pick up the pace a little. The "Hello World" extension didn't use a lick of JavaScript (apart from the JSON), and we didn't actually write any HTML, either. This section will rectify that.

First, let's take a look at the new and improvedmanifest.jsonwe'll use for this extension:

{

"name": "RSS Fetcher",

"version": "0.1",

"description": "Keep up with the latest from [insert web site here].",

"icons": { "16" : "images/icon.png",

"48" : "images/48.png",

"128" : "images/128.png"},

"homepage\_url": "http://insert\_web\_site\_here.com/",

"background\_page": "background.html",

"permissions": [

"http://insert\_base\_rss\_url\_here.com/\*"

],

"browser\_action": {

"default\_icon": "images/icon.png",

"default\_title": "[INSERT WEB SITE NAME HERE] Fetcher",

"default\_popup": "popup.html"

}

}

Let's quickly walk through what's new here.

* First, you'll notice I've addedicons. These are the icons that will display as the extension's icon in various places, including the Extensions page of your browser and in the Chrome Web Store should you decide to distribute your work.
* homepage\_urlpoints to whatever site you want to associate with the extension you've written.
* background\_pageis an HTML page that will run in the background constantly, allowing you to maintain a certain state or perform regular actions for different parts of your extension. In our extension, the background page will poll the RSS feed to see if anything new has been pushed to the feed.
* permissionstells Chrome what special permissions you need access to. Chrome sandboxes extensions so they don't have access to all your browsing activity unless they request it (and you allow it). The permissions property allows you to set exactly what permissions you need. In our case, we need permission to load an RSS feed from a root URL, and we want to be able to create HTML 5 notifications when new items arrive.

You'll have noticed that ourmanifest.jsonfile also calls for a lot of new files and folders, so instead of requiring you to set them all up yourself, you can download a basic, working version of this code that fetches from the Lifehacker RSS feed [here](http://adampash.com/s/RSS%20Fetcher.zip).

(I've decided to include the images I used in the Lifehacker Notifier extension, but obviously you can swap out whatever images work for you.)

You'll also notice that I've included jQuery. If you're unfamiliar with jQuery, it's a JavaScript framework that makes doing a lot of things in JavaScript insanely easier. If you do anything on the web with JavaScript and you're not interesting in rebuilding the wheel and you are interested in saving a lot of time, you should learn to use jQuery. (Also, lucky you! The very cool web site Codeacademy just released [an introductory guide to jQuery](http://www.codecademy.com/courses/jquery-and-the-dom).)

This all may seem like it got complicated really quickly, but the main difference between our Hello World extension and this more complex RSS Fetcher is actually quite simple: Ourpopup.htmlfile will now contain HTML and some JavaScript. Together, they'll *actually do things* beyond displaying "Hello World!" every time you click the button.

The other thing sets Chrome extension development apart from your everyday HTML and JavaScript is the [Chrome extension APIs](http://code.google.com/chrome/extensions/api_index.html), which provide access to all kinds of functions that blur the line between your extension and the browser. So let's try a few basics.

**How Our New Popup.html Works**

The newpopup.htmlfile in our RSS fetching extension works like this: When you click the extension's button, it loadspopup.html. When that loads, it passes a call tobackground.html, asking it to fetch our RSS feed. Oncebackground.htmlhas successfully fetched the feed, it passes the feed's XML back topopup.html, which then parses the feed into some friendly HTML, which it then displays inside the popup.

**Message Passing Between Background.html and Popup.html**

Because of various security sandboxing going on in Chrome, some parts of your extension can't access the same APIs or information that other parts of your extension can. To get around this, one common technique you'll need to employ in your Chrome extensions involves [passing messages and data back and forth between different parts of your extension](http://code.google.com/chrome/extensions/messaging.html). To demonstrate the basics of how this works, I've put the method that fetches the RSS feed intobackground.html, and I call it frompopup.html. Inbackground.html, you'll notice thefetch\_feedmethod:

function fetch\_feed(url, callback) {

var xhr = new XMLHttpRequest();

xhr.onreadystatechange = function(data) {

if (xhr.readyState == 4) {

if (xhr.status == 200) {

var data = xhr.responseText;

callback(data);

} else {

callback(null);

}

}

}

// Note that any URL fetched here must be matched by a permission in

// the manifest.json file!

xhr.open('GET', url, true);

xhr.send();

}

It's a very basic function that takes a URL and a callback function as parameters, fetches the feed, then passes it to the callback function. To listen to the other parts of your extension, we add this code:

function onRequest(request, sender, callback) {

if (request.action == 'fetch\_feed') {

fetch\_feed(request.url, callback);

}

}

// Wire up the listener.

chrome.extension.onRequest.addListener(onRequest);

The last line, which uses the wonderfully useful [chrome.extension API](http://code.google.com/chrome/extensions/extension.html), tells the page to listen to requests from other parts of the extension. When it receives one, it's going to pass it to theonRequestfunction. As you can see, theonRequestfunction then checks to see if the request is asking for an action that it knows how to handle (in this case, a request to fetch a feed), and if it does, it calls that function.

Now let's jump over to thepopup.htmlfile. When it loads, the page is entirely empty. This is what kicks off changing that:

$(document).ready(function() {

fetch\_feed();

});

Ourfetch\_feedmethod then fires off this request:

function fetch\_feed() {

chrome.extension.sendRequest({'action' : 'fetch\_feed', 'url' : 'http://lifehacker.com/index.xml'},

function(response) {

display\_stories(response);

}

);

}

As you might have guessed,chrome.extension.sendRequestsends out the request to other parts of your extension that may be listening. And as we know,background.htmlis listening for this exact request! So our request bounces frompopup.htmltobackground.html, which fetches the feed, then passes it *back* topopup.html, which then calls a function calleddisplay\_stories, passing with it the response frombackground.html. The display stories function (which I won't step through here but which you can see in the source) then uses a little jQuery and JavaScript to parse the XML feed and render the stories in the popup.

***Do you see how unbelievably easy this is???***

I kid. Once you get the hang of it, though, making Chrome extensions actually is very easy, and very fun. (Obviously the complexity varies depending on what you want to make.) If you're comfortable with HTML and JavaScript (oh, and I guess CSS if you want to make it pretty), you can do so much, and the learning curve is pretty gentle. So go forth, intrepid coders, and make thee some awesome extensions!

**Troubleshooting**

You're great at everything (go you!), but you're going to stumble from time to time, particularly with your JavaScript. You'll find two invaluable（无价的） tools to help debug. The first isconsole.log, which is a JavaScript method that writes text, objects, or whatever you pass it to Chrome's JavaScript console so you can catch where things may have gone wrong.

The second is Chrome's awesome Developer Tools—and more specifically, the JavaScript console and DOM inspector. You can pull up the console for your popup by right-clicking your extension's button and selecting Inspect popup. Likewise, you can pull up the Developer Tools forbackground.htmlby clicking thebackground.htmllink in the expanded view of the extension in <chrome://settings/extensions>.

**Helpful Resources**

All of my Chrome extension-building knowledge comes from Google's very thorough documentation and good ol' fashioned World Wide Web searches when I'd get stuck. So you should certainly check out:

* [The Official Google Chrome Extensions Developer's Guide](http://code.google.com/chrome/extensions/devguide.html): It's the starting point for everything you need to know, lays our all the various extension APIs you may want to use, and so on. If you know what kind of extension you want to make, I'd highly recommend taking an hour or so, reading through the docs, and taking notes/saving links whenever you stumble upon something that sounds like you'll need it to accomplish your extension goal.
* [Stack Overflow's Google Chrome Extension tag page](http://stackoverflow.com/questions/tagged/google-chrome-extension) is filled with great Q&As regarding Chrome extension development. If you're stumped by something, give StackOverflow a search, give the web at large a search, and if you can't find the answer, try posting your question to Stack Overflow. It's awesome.

**Get Coding**

So, enough of this starter guide. Time for you to roll up your sleeves and do some extension development of your own. If you've got any questions, or just want to share what extensions you're planning to make, let's hear it in the comments.