Photo courtesy of trippChicago and licensed under Creative Commons

Rituals and Chrome Extensions

Let's talk about rituals.

Now I'm not talking about religious rituals, official services, or anything so solemn. I'm talking about the everyday type that you don't even think about. For example, you probably have a standard set of activities you do every morning—get up, shower, eat breakfast, brush your teeth—fairly standard stuff. Mundane as those activities may seem, that repeated set of behaviors constitutes a ritual—one which you perform every morning.

Now, you're probably asking yourself, "why is this guy talking about rituals involving toothbrushes in a blog about technology?" Admittedly, that's a fair question.

One of my ever-present goals is to make life as efficient as possible—even (especially!) if that efficiency takes vastly more effort to implement than it saves. My latest additions to that basket of life-efficiency-enhancers have stemmed from identifying inefficiencies in my rituals, and finding ways to streamline them.

Good morning!

Each morning I get up and go through my morning routine. Near the end (around 6:50am), I check a Chicago Transit Authority (CTA) Bus Tracker application on my phone to see when the next bus will arrive. I want to leave the apartment when the next bus is 2–4 minutes away to minimize time waiting idly at the bus stop, while leaving enough of a buffer that I don't have to run to catch it. If the bus is predicted to arrive outside that window, I proceed to mash the refresh button on my phone until it gives me a number I like. Unlocking my phone, opening the app, selecting the stop, requesting data, and then potentially repeating that procedure is not an efficient way to determine when I should leave.

Cue lightbulb number 1—a Chrome extension to tell me when to leave.

I started my foray into Chrome extensions about 6 months ago with a simple one that added an option to the context menu to identify an Instagram photo's URL. It was about as basic as it could be, with no interface elements besides the single context menu option—but it exposed me to what was possible with the SDK, and taught me the process to build and deploy something to the Chrome store. I had also toyed with the CTA API a while back while looking into building an Amazon Echo application for CTA bus tracking (which never panned out)—which gave me both pieces I needed to build an extension to tell me when to leave.

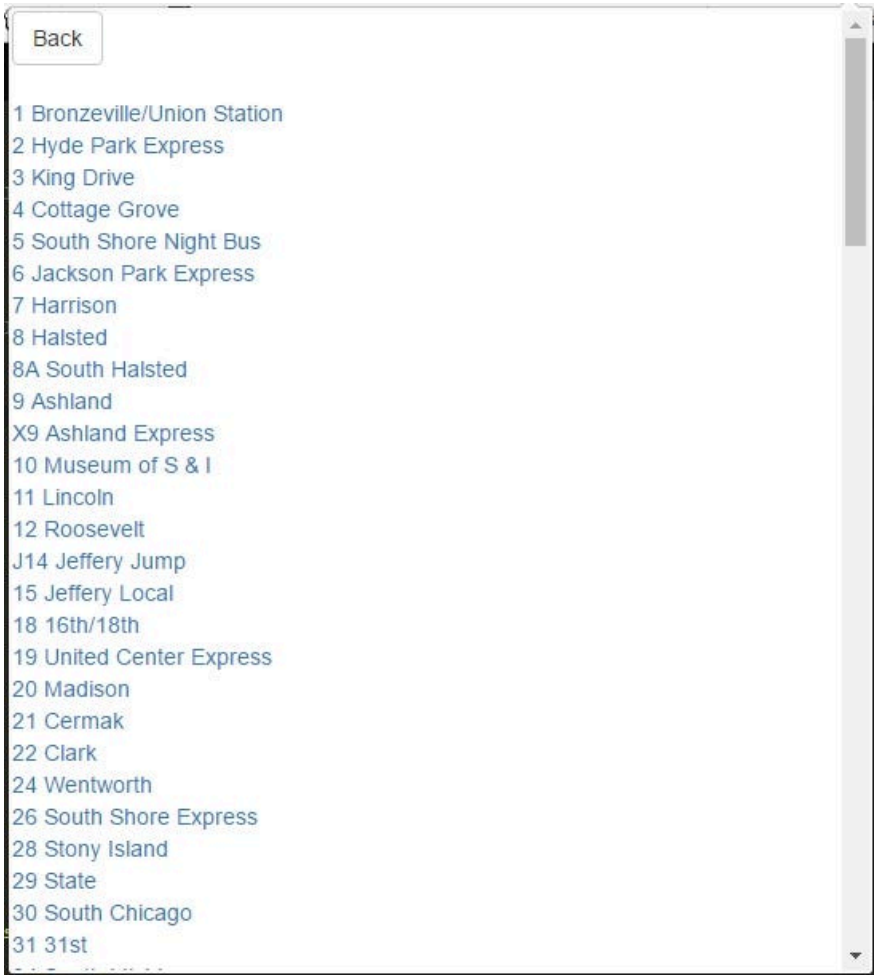
Route planning

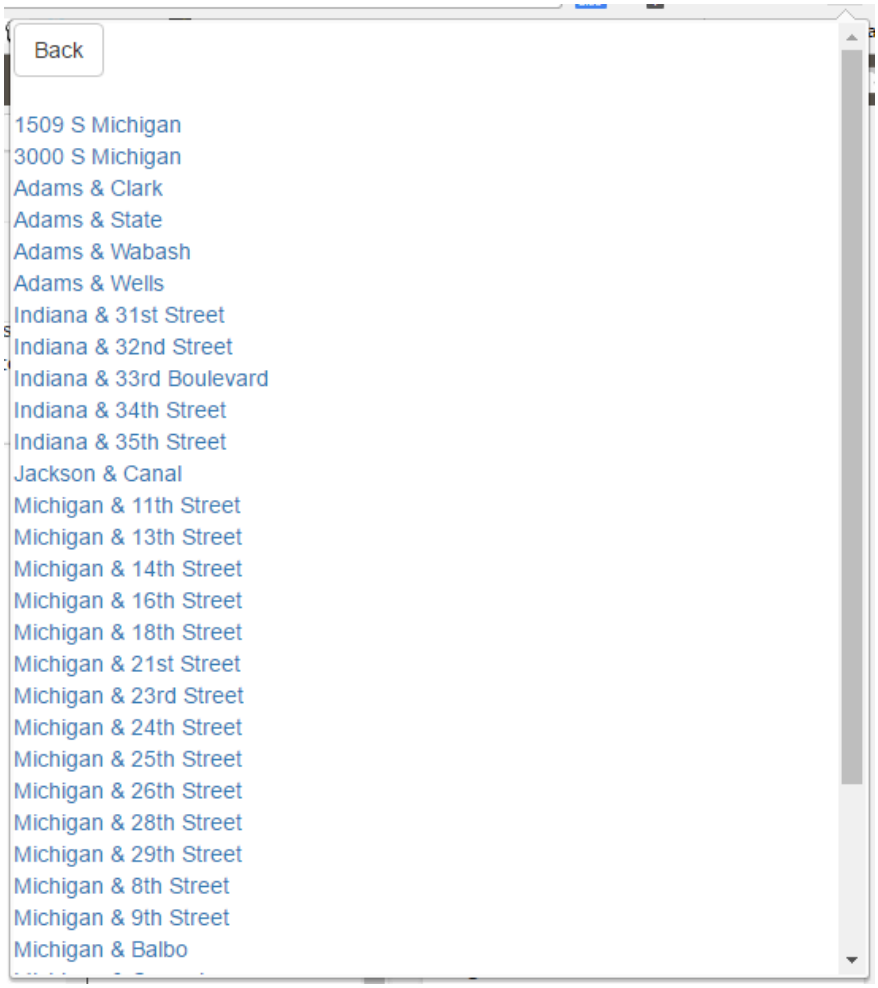
First, I planned out what the app should do. There would be 2 components—one for selecting the stop to track (the user would initiate tracking by clicking on one of a list of 'pinned' stops, and have the ability to add new pinned stops), and one for displaying the status of the selected stop. While tracking, I want to know how long

until the next bus comes, how fresh the data is, what bus line I'm tracking, and when one is almost here.

For the first component, I needed a pretty big working area—there are well over 100 CTA routes I'd need to show. To support this, Chrome's Extension SDK allows you to specify a web page that's displayed as a 'Popup' when you click on the application's icon. From that page, you can execute a javascript method on the background JS using the Chrome Extension SDK's getBackgroundPage function—allowing the popup UI to call a method to start a timer on the background to update the icon

Building out the interface to select a stop was quite simple. The BusTracker API is really well designed; it's built in such a way that you can get a list of all bus lines, directions, and stops—each based on the result of the previous. Once you have the stop ID, you can request 'predictions' about when the next bus will arrive there. It was simple to build out a series of screens to allow selecting a stop, then persisting the saved stop IDs to the Chrome synchronized storage using the Extension SDK.


Back
1 Bronzeville/Union Station
2 Hyde Park Express
3 King Drive
4 Cottage Grove
5 South Shore Night Bus
6 Jackson Park Express
7 Harrison
8 Halsted
8A South Halsted
9 Ashland
X9 Ashland Express
10 Museum of S & I
11 Lincoln
12 Roosevelt
J14 Jeffery Jump
15 Jeffery Local
18 16th/18th
19 United Center Express
20 Madison
21 Cermak
22 Clark
24 Wentworth
26 South Shore Express
28 Stony Island
29 State
30 South Chicago
31 31st



Getting iconic

For the second component, I knew that I wanted the totality of the interface required for actively tracking state to be shown through the icon at the top of the browser window—having to open a new pane or click on something after tracking has been started would defeat the whole purpose of the project. The challenge, then, was how to fit everything into a 19 pixels square area—a ridiculously small amount of space. For context, that's roughly the amount of space that 4 letters on this page (in a 2×2 character square) would take up.

Which, at actual size, looks like this

After much mucking about in a 19x19px box in Paint.NET, I came up with a design that covered what I needed. The background would be a bus, with a 13x7px windshield. Using a font that took just 3x5px per number/letter (based on http://www.sum-it.nl/en200351.html), I would then draw the bus line number inside the windshield of the background. The 'freshness' of the data (or the time until the next refresh) would be represented by 6 4.4px dots overlaid at the top of the icon, which fit perfectly into the 19px width, with 1px between each dot and on either side. When the bus got close, I would turn the background of the bus's headlights green (time to go!)—and I would use the icon's Badge text (generally used for notifications) to indicate the number of minutes until the next arrival.

Great! Now… how do I actually make that happen?

Back in the extension's background javascript, the Extension SDK allows you to set the icon for the application one of two ways—either by referencing a static image file, or by building the icon in an HTML5 Canvas element, and requesting the icon be loaded from a section of that Canvas. Clearly, to support the notifications being inside the icon, I'd have to build it up using the latter.

To start, on the canvas, if the headlights were supposed to be green, I drew green squares in the spot where the headlights would end up. I then loaded in a background image of the bus, which had transparent headlights and background (but not windshield—we want a light background for the numbers). If the headlights needed to be green, the square I drew in initially would show through the transparent spots in the Bus image. I created image files for each digit and letter that was needed (all 3x5px), and named them appropriately. Based on the number of digits in the bus line (151, 22, etc), I calculated the pixel position of each character, and loaded the appropriate images onto the canvas in the appropriate spot. I then drew little circles along the top to indicate when data will be refreshed, and set the overlay text to the number of minutes until the next predicted arrival. Once that was all complete, I requested that the square of the Canvas into which I'd been drawing be loaded as the Extension's icon—and that was that!

Again, at actual size

Select a pinned stop, and the icon will start refreshing automatically for 10 minutes, with all the information you need visible at a glance. Efficient!

The extension is available for free in the Chrome store

Efficiency number one—attained. Stay tuned for part 2!

permalink

. . .

Originally published at blogs.claritycon.com on October 10, 2016.