Extension Options V2

kalman + ericzeng, Q2 2014.
(Updated on 8/28 to reflect current implementation for launch review)
bug

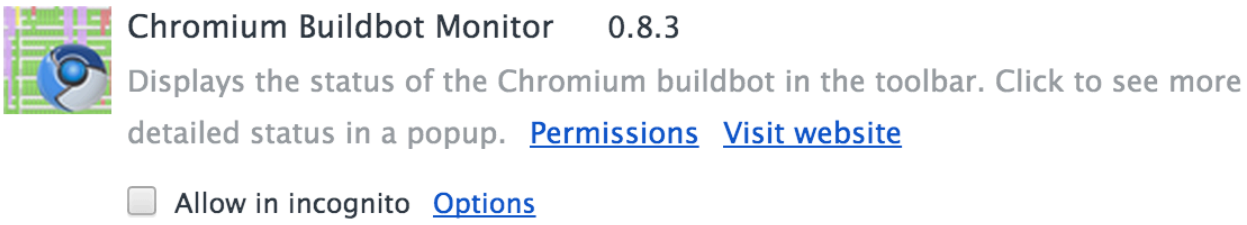
Background

Extensions can have options pages. They are declared in the extension's manifest:

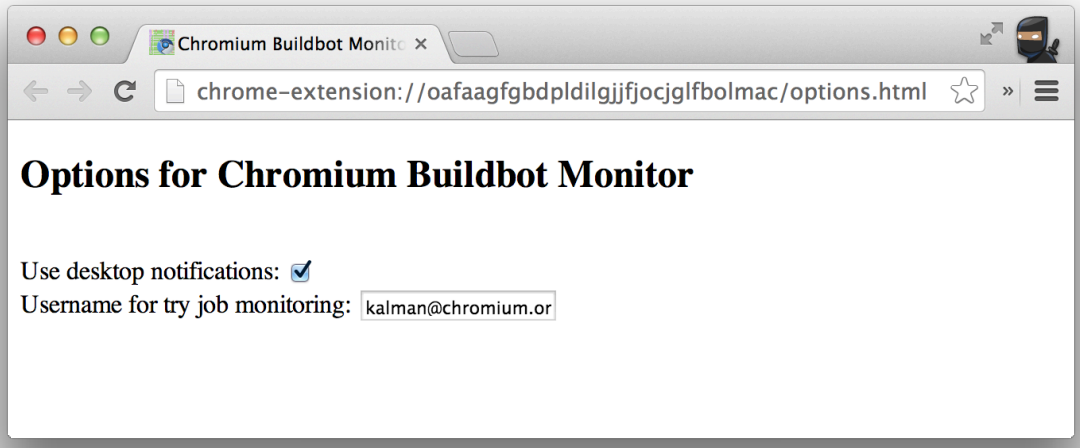
```
{
  "name": "My extension",
  ...
```

```
  "options_page": "options.html",
  ...
}
```

which renders a link on the chrome://extensions page:



which when clicked opens that page in a new tab:



in this case a particularly unattractive one, and note the URL, and imagine it being in a tab that isn't artificially small.

Generally speaking extension options pages are written in 3 varieties: unstyled (as above), styled in their own manner, or styled by copying Chromium's CSS. They rarely look good by themselves, are inconsistent with each other, and even those which do look good are at continual risk of regressing as Chrome styling changes.

Furthermore, often they're quite minimal yet are required to fill a full tab, with a user-unfriendly chrome-extension://hnnpdpdghppcedbfaiceoemakedlegla/options.html URL at the top.

See Appendix for a bunch of examples.

Fixing this will improve the user experience of Chrome (it's glaringly unpolished), reduce developer overhead for making options pages, and until this is fixed it's embarrassing to suggest that we make power user features controllable by Extension APIs.

Extensions *are* part of the browser. They're an extension of its functionality. Their options should be given as much treatment as the way we let extensions create toolbar buttons (for example).

Some references:

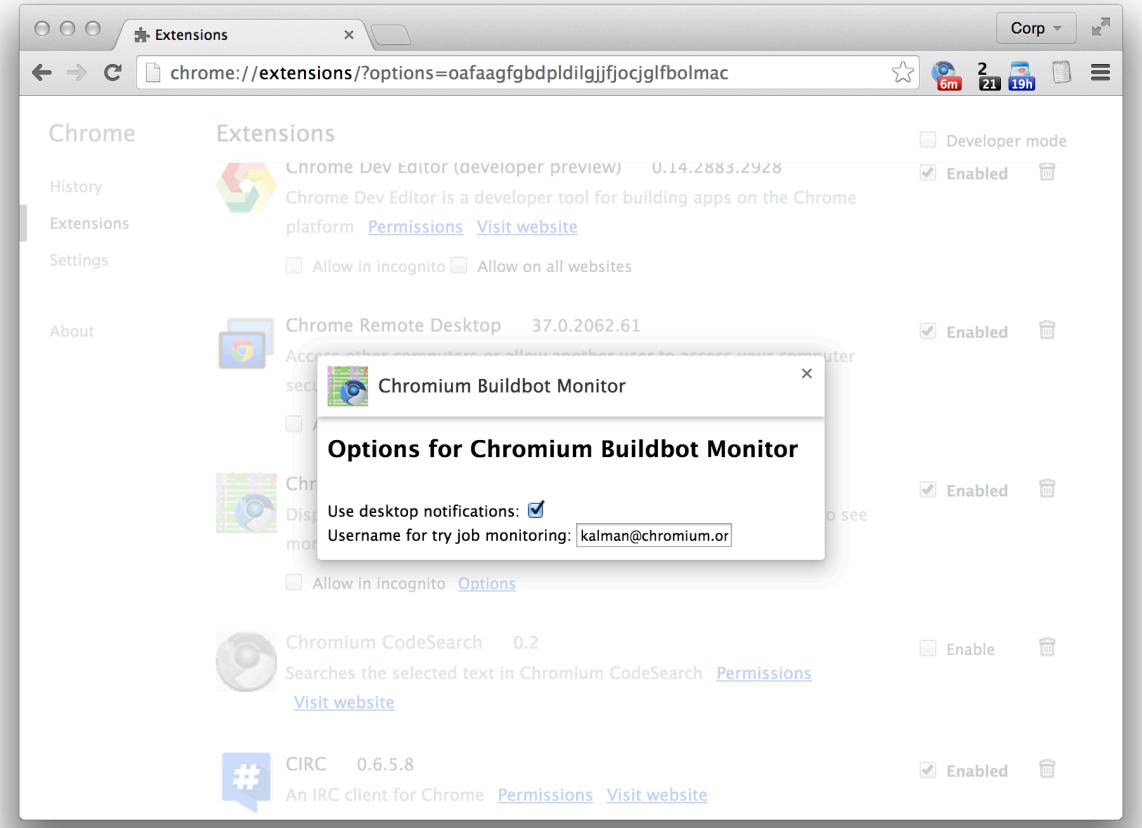
- The original crbug.com/25317 has many comments and 908 stars (the 9th most starred bug in the whole Chromium bug tracker).
- crbug.com/319429 recommends writing an extension with the fontSettings API to control a power user / accessibility setting.
- crbug.com/72021 debates showing URLs for options pages. This would be moot if they were displayed as dialogues.
    - Apparently there was similar discussion for crbug.com/219446 but it was resolved over email.
- Stack Overflow question: here.

Goals

1. Extension options should feel like an integral part of Chrome's settings.
2. Extension options should be styled like Chrome, *and stay up-to-date as Chrome changes*.

Implementation

Before, and after (and you can try this yourself with this flag; Canary channel most likely to succeed):



Note that Chrome styling hasn't been implemented yet; it will look even better after that, and *even better* once the author of Chrome Buildbot Monitor removes its title.

Options should be an integral part of Chrome

Proposal: create a new element <extensionoptions> tag based on the same out-of-process technology that <webview> and <appview> use. This will load the extension's options page *in the extension's isolated process* ("guest") and provide the embedder with the necessary APIs for it to be well integrated.

- Difference to <webview>: webview can only load web content and does so in isolated site instances. <extensionoptions> will load only extension/app pages within their default site instance.
- Difference to <appview>: appview provides a rich API to the guest, including that it knows when it's being embedded, the ability to reject embedding, communication with the embedder, and so on - generally cooperative behaviour between the guest and embedder. We want the options pages to be all-but-unaware of how they're being used.

Note that we call this <extensionoptions> not <extension-options> because the former is an element name that needs to be whitelisted, so won't conflict with anything in the wild.

Embedding in WebUI

The options pages are shown in the same way that dialogues are opened in the current chrome://settings UI. For example, the "Advanced sync settings…" button[1] + dialogue.

For us this means rendering the <extensionoptions> element to determine its initial size (this is determined via an API), attaching that to a WebUI dialogue, then showing it. In most cases this just works (exceptions are discussed later).

Security

As mentioned above, we have process-level security by hosting the content in the extension's process not the WebUI process. We have UX-level security because it's clearly attributed to the extension. Note that security interstitials only affect the options content (good).

Opt-in/out

Initially we'll need an opt-out mechanism for this because we've found not all options pages work. Some rely on the page being in a tab (e.g. by using chrome.tabs.getCurrent() or relying on message senders having tab IDs), others don't work for mysterious reasons which we can't determine - possibly asynchronous UI frameworks - we hope that given ample time in dev channel and the ability to opt-out as a last resort will solve this.

We considered opt-in but given we eventually plan on removing any opting this too soft a stance.

Attribution

Dialogues are clearly attributed to the extension whose options page they represent using its icon, name, and clear delineation between trusted (WebUI) and untrusted (the extension's) UI. This solves a couple of issues:

- Extensions could phish parts of the UI like the signin page.
- It would look odd for the extensions options pages - which initially certainly won't look much like WebUI - to appear like they're blessed in Chrome. I was actually most worried by this initially, but it looks totally fine.

A second class of attribution issues are that browser level UI like infobars and alerts may appear to come from the WebUI pages, not the extension. A similar issue is that there will no longer be any way for the options pages to initiate these UIs. I've filed bugs 424635 and 424630 for why this doesn't matter (and in fact, is desirable).

Navigation

There are 2 tricky parts of navigation: when and how to navigate to these pages, and when and how to navigate out of them.

---

[1] Button UI not implemented; still looks like a link.

Navigating in: we change the "Options" link to be an "Options…" button to imply that it opens up a dialogue, not a separate tab or full-page navigation. We also change the "Options" context menu item on the extension's toolbar button to open up the settings page with the dialogue pre-opened.

Going directly to the chrome-extension://.../options.html page will continue to open in a tab. Intercepting navigations is harder than trivial, makes debugging harder, and I don't think it would be the right behaviour anyway.

Navigating out:

- Links that want to be opened in a new window/tab will continue to do so (target=_blank, window.open(), user choosing "open in new tab" for example).
- Other links that are to the same chrome-extension://… domain will continue to open within the <extensionoptions> element.
- Other links that are not in the same domain will open in new windows[2].

Options should be styled like Chrome

Proposal: write a new user agent stylesheet, similar to platform_app.css, and give extensions the ability to apply this to their options pages. This stylesheet should be based on WebUI's widgets.css, minus assumptions about DOM structure - such as the "custom_appearance" class.

This stylesheet will need to be kept in sync with the real widgets.css by the extensions team. We intentionally didn't share the same stylesheet.

Opt-in/out

The styling will need to be opt-*in*, it's highly likely to break options pages in subtle ways to always apply it. I initially intended to make this opt-out; but I'm not too worried about this now that it turns out even unstyled options pages look pretty decent once they're embedded.

Navigation

Extensions sometimes implement their options pages across multiple pages. Since we'll be navigating in-frame for such navigations we'll need to continue to apply this CSS to subsequent page loads.

Manifest changes

To implement opt-in/out and maintain forwards compatibility (not to mention backwards compatibility) we will introduce a new manifest object options_ui.

```
{
  "name": "My extension with options",
  ...,
  // Stable/beta channels of Chrome will continue to read
```

[2] Not fully implemented, some types of navigation don't work.

```
  // the options_page key. An extension could use this to
  // provide a different options page in here and options_ui.
  "options_page": "options.html",
  // Dev and canary versions will read the options_ui key if
  // it's available (and fall back to options_page of course).
  "options_ui": {
    "page": "options.html",
    // Opt-in to Chrome styling (default false).
    "chrome_style": true,
    // Opt-out of opening in a dialogue (default false).
    // This key will eventually be ignored.
    "open_in_tab": true,
  },
  ...
}
```

Future of settings embedding

The future involves (at least) two changes in this area:

- Chrome settings are moving to an app. On ChromeOS in particular it's odd that settings are exposed in a Chrome tab, so that's being fixed.
- Material design.

We're prepared, and in fact we can adapt neatly and still fulfil the goals of this project:

1. Write a material design (polymer) shared module.
2. Make these settings apps used that shared module.
3. Add a new key in options_ui to include the shared module automatically.

The settings app(s) can continue to embed extension options because the <extensionoptions> element can be embedded anywhere. The chrome_style key and CSS becomes somewhat moot but the Chrome styles still look better than the system defaults, so, not entirely.

When the time comes, this will likely require a more formal design process than just a paragraph.

Other possible implementations for style injection

Provide an entirely declarative API

A lot of discussion went into whether we should tackle this problem from a much higher level - invent a language for describing options pages (a JSON dialect for example) and generating options pages from that. Everything that used it would look fantastic.

However it's increasingly complex to cover all use cases, I think impossible (see the variety of options pages we have). Having a stark contrast between generated pages and hand-written

ones would look bad. If we can't give every extension options page the chance to benefit from Chrome styles then it's less appealing to enforce embedding.

I do hope that given the tools to make rich options pages we'll start to see frameworks, even integration with data binding and other frameworks. That is something we can't offer.

It's a natural next step in this project, in fact, to provide that sort of thing in the Chrome Dev Editor.

Expose a set of custom elements

A form of this is discussed in the long-term future of embedding, but this was also considered in the near term, and we decided not to go with this approach.

This is more complex than what we need, and is overhead for developers to need to learn some other set of HTML tags when input tags work just fine at the moment, and we're not quite ready for the future (nor I do I believe that building these into Chrome is the simplest approach).

Expose a single CSS resource on a resource URL to all extensions

The initial plan was actually to add a new safe resource URL - such as chrome-resources://... - which solves the styling in a neat web-idiomatic way, and easy to opt into. The main reason we didn't go with this is to avoid the proliferation of special Chrome protocols.

This would have let WebUI and extensions read from exactly the same CSS stylesheet, but this was avoided so WebUI developers wouldn't need to worry about breaking extensions. That worry can be the extension team's.

Don't programmatically expose any resources

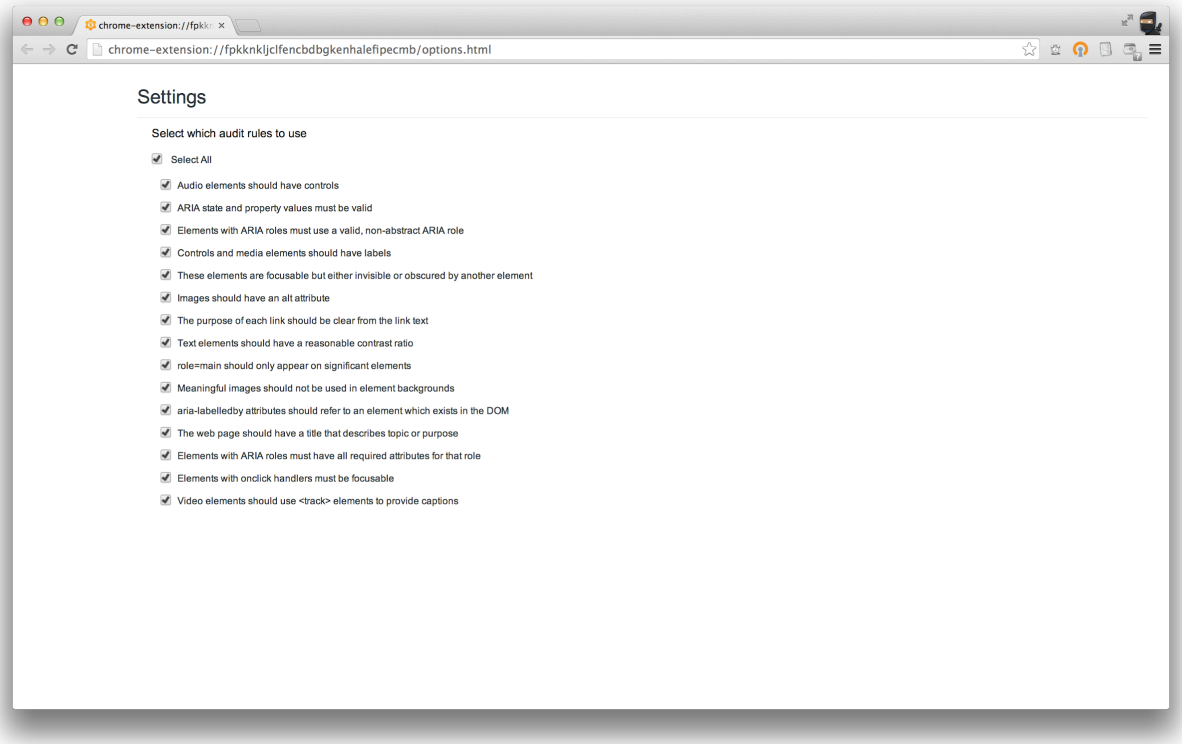
Lastly, a suggestion had been to publish stylesheets separately from Chrome (e.g. on developer.chrome.com) so that developers can include them however they like.

This is quite flexible, but there's a lot of overhead here. It also doesn't give us the chance to change styles over time and have extensions update. I don't want to consider the license issues.

Appendix

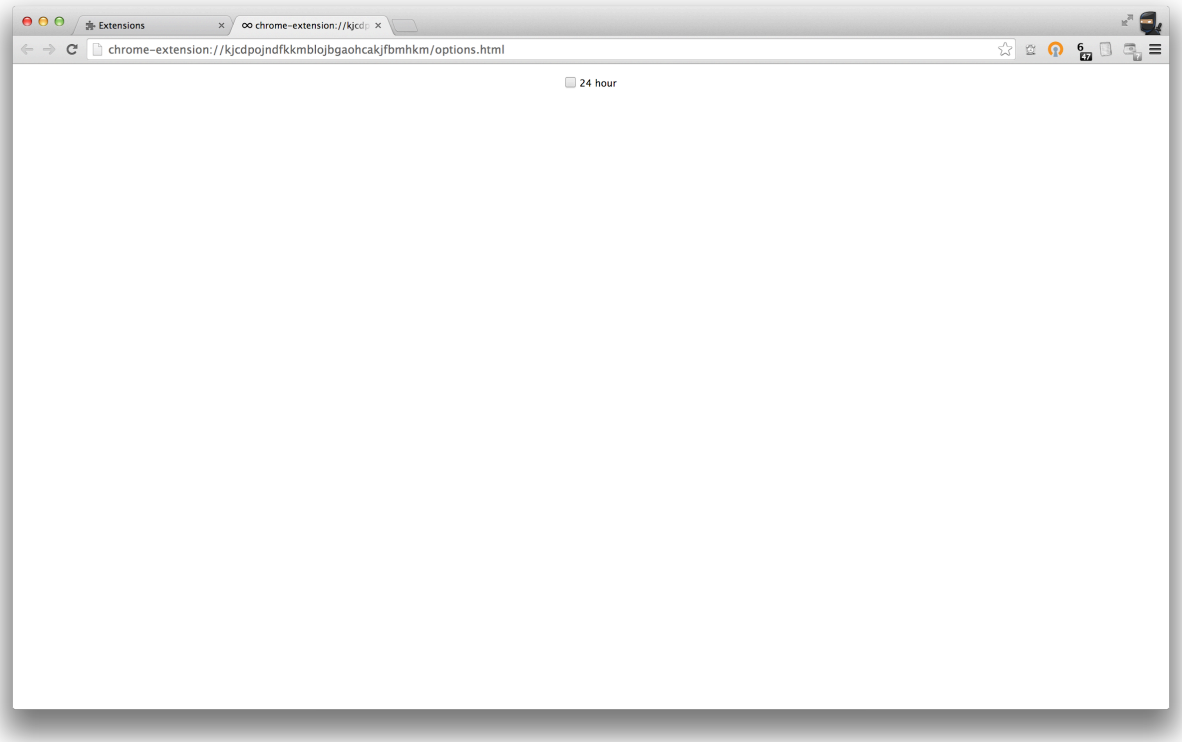
Accessibility Developer Tools

Good looking, took the Chromium styles, but note things like the URL bar and tab-ness. The mock above looks much better.



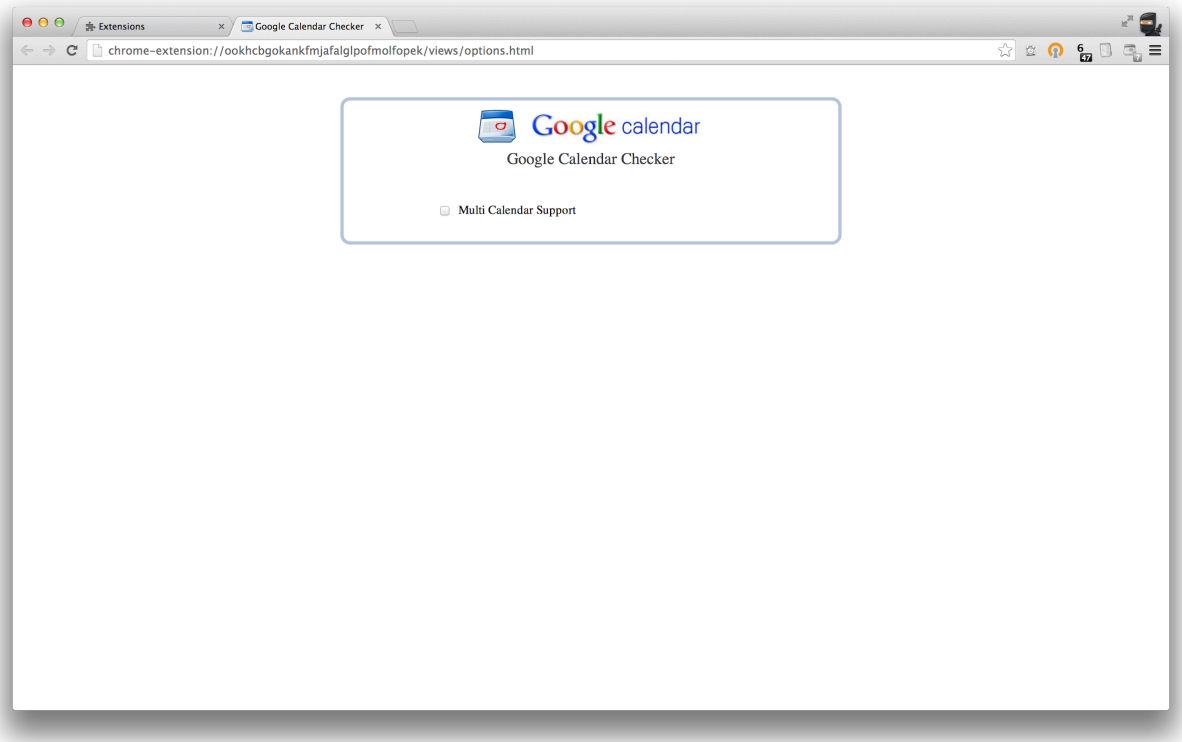
Browser Clock

One of my extensions. A tiny options page, a lot of whitespace.



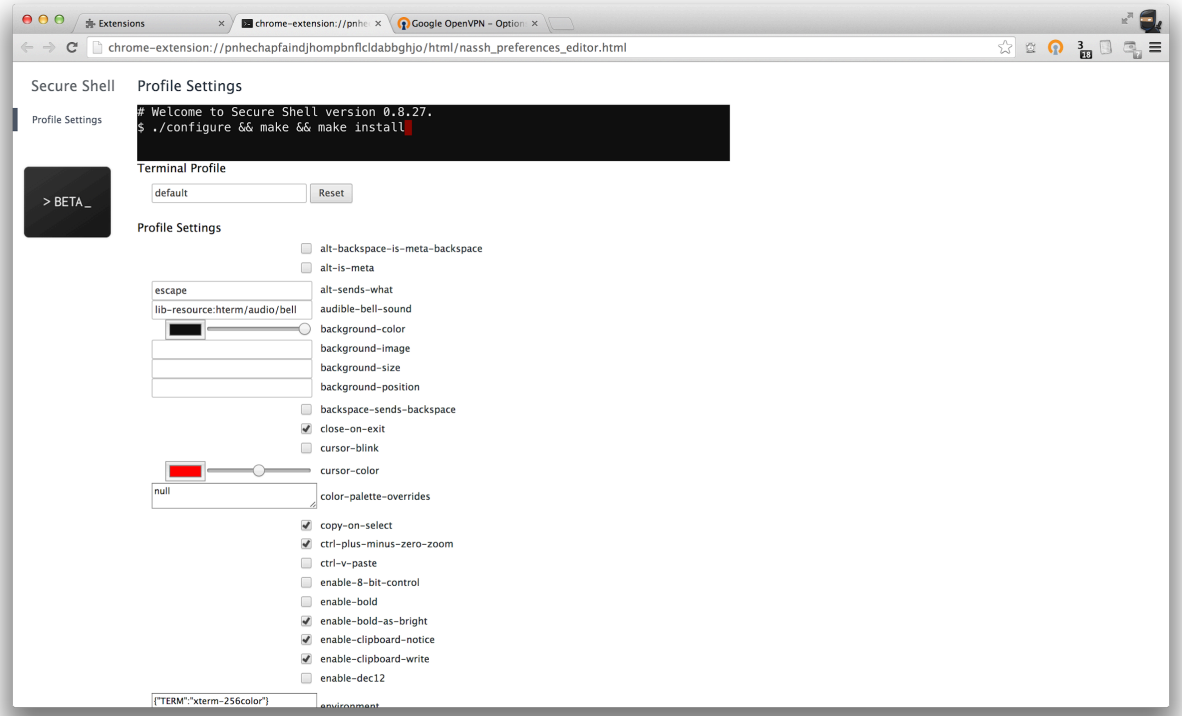
Google Calendar

Not good looking, and it's a "by Google" extension.



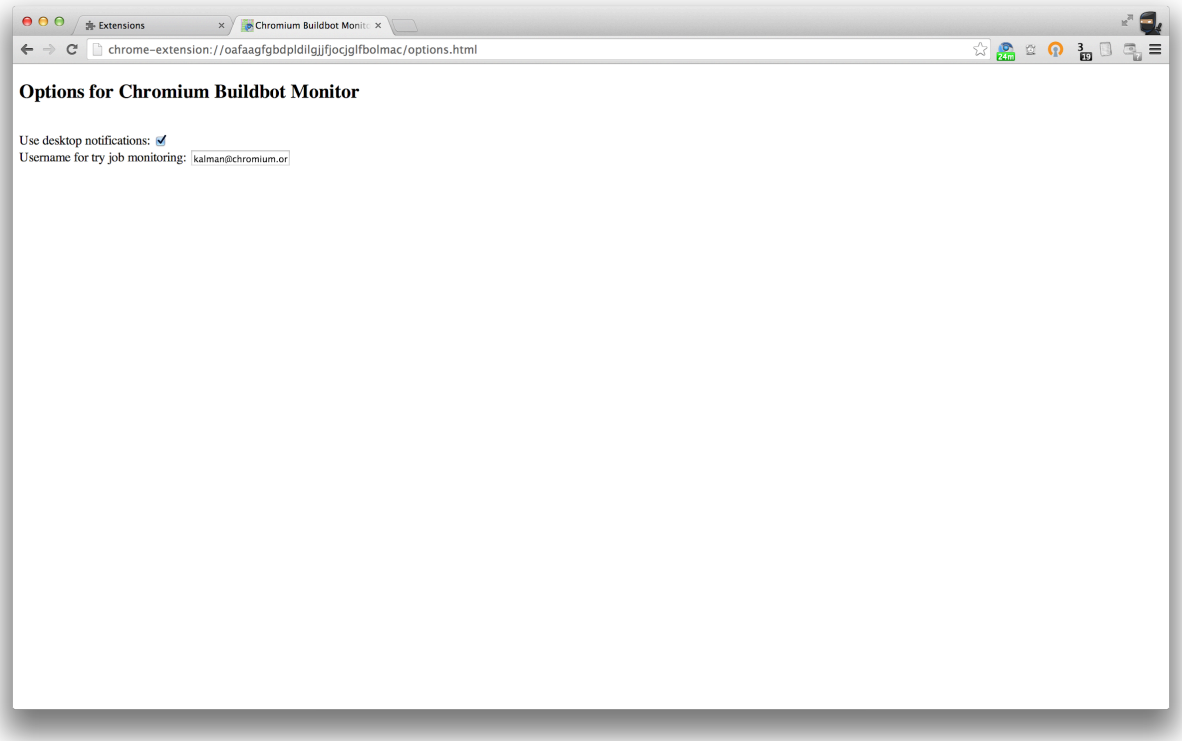
Secure Shell

A big options page so we'll need scrolling… looks like it's already taken the Chrome styles).



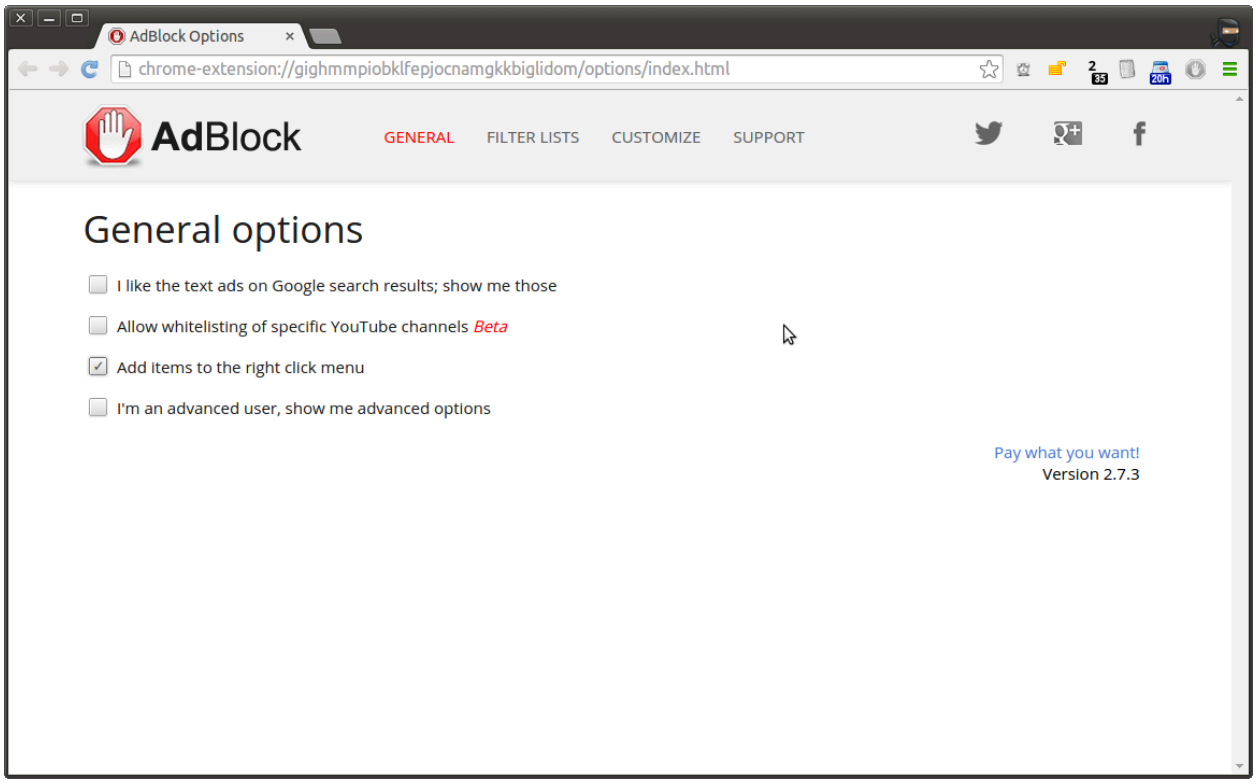
Buildbot Monitor

A plain options page, perfect candidate for this work.



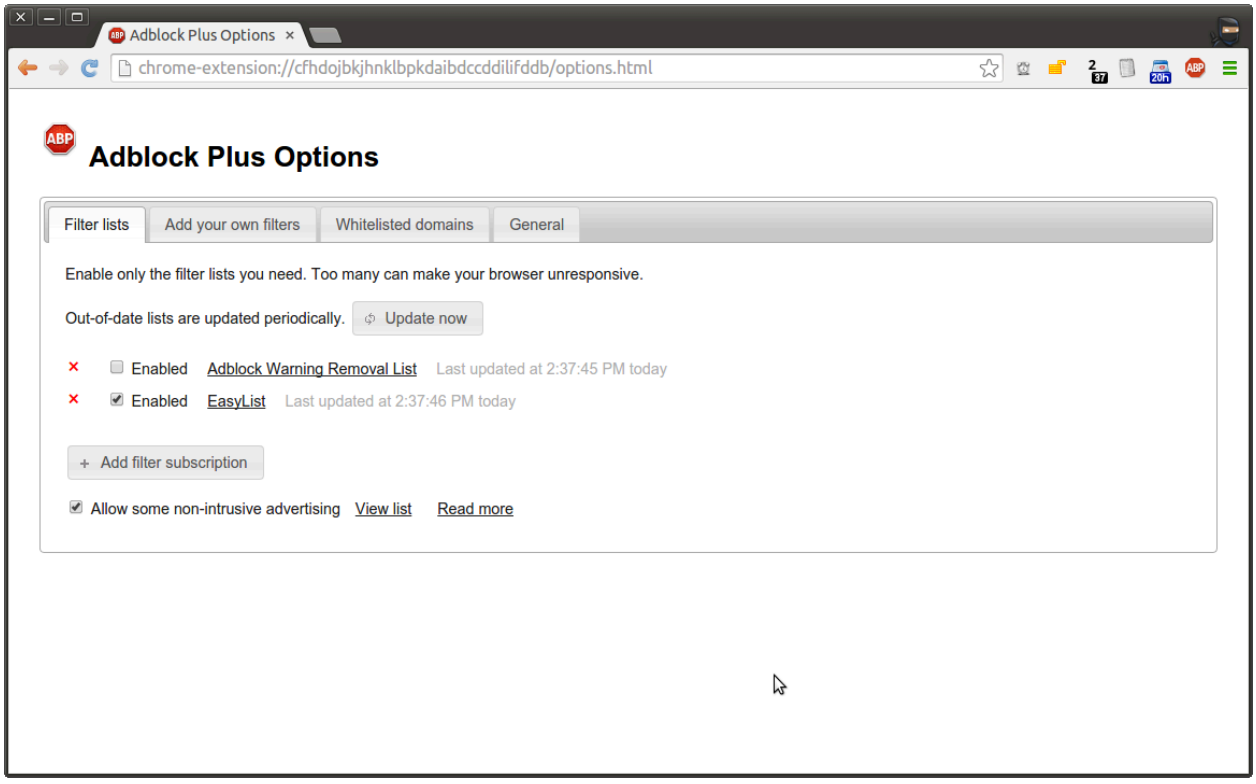
Adblock

The most popular options page in the world.



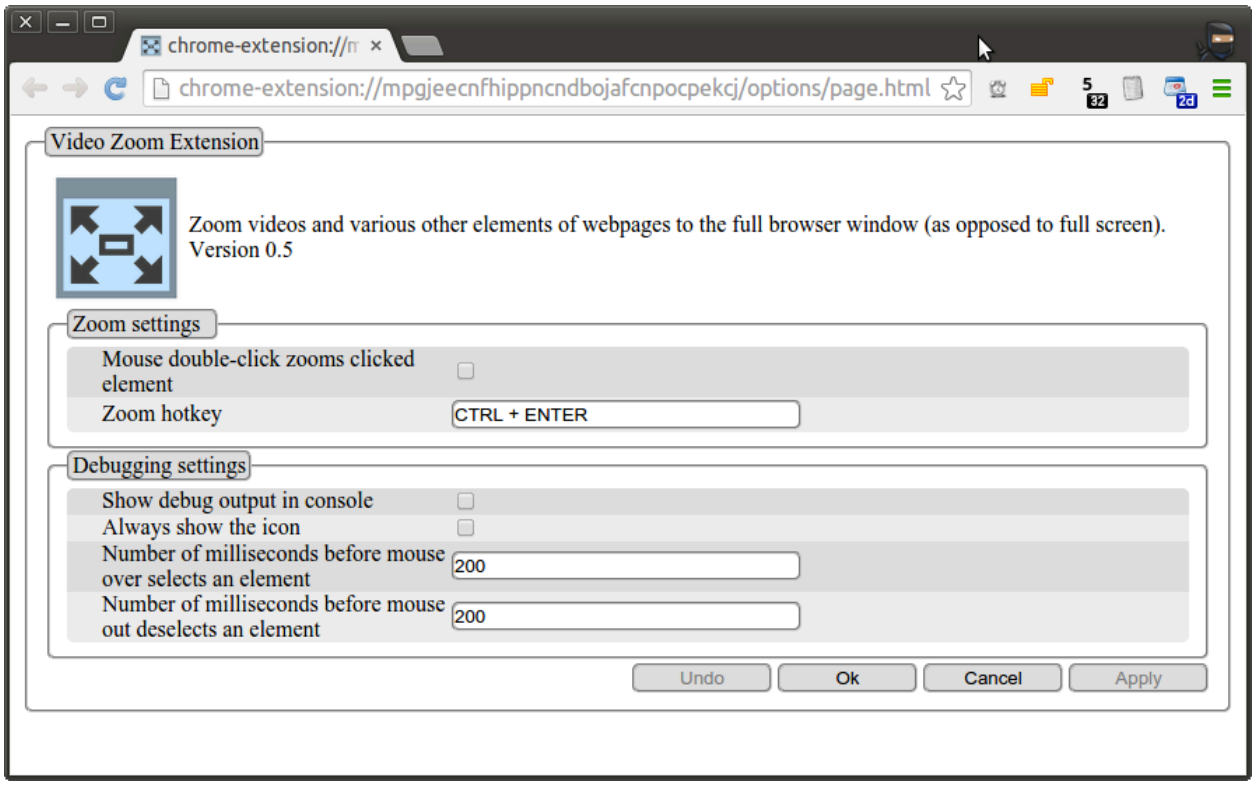
Adblock Plus

Which would be the second most popular options page in the world.

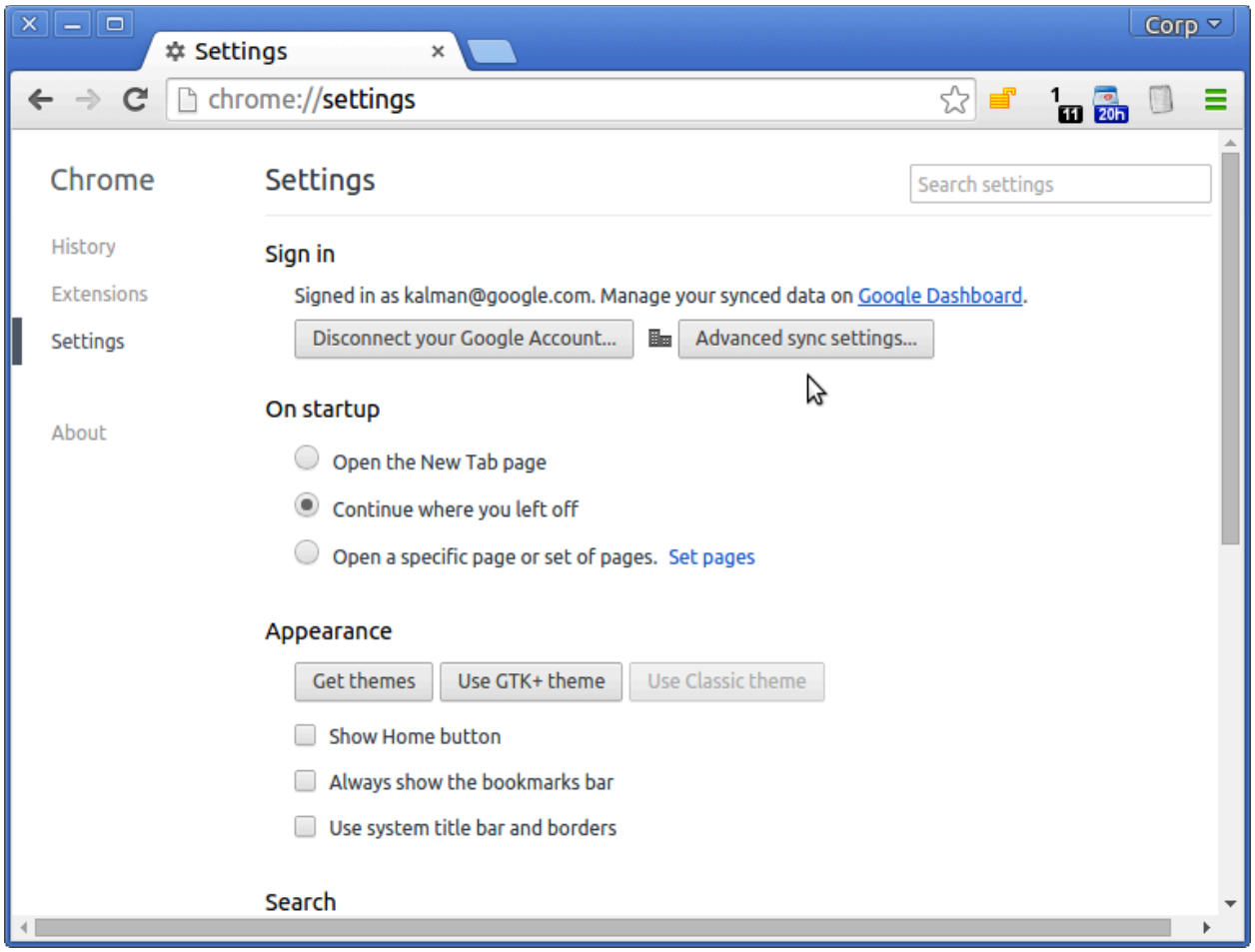


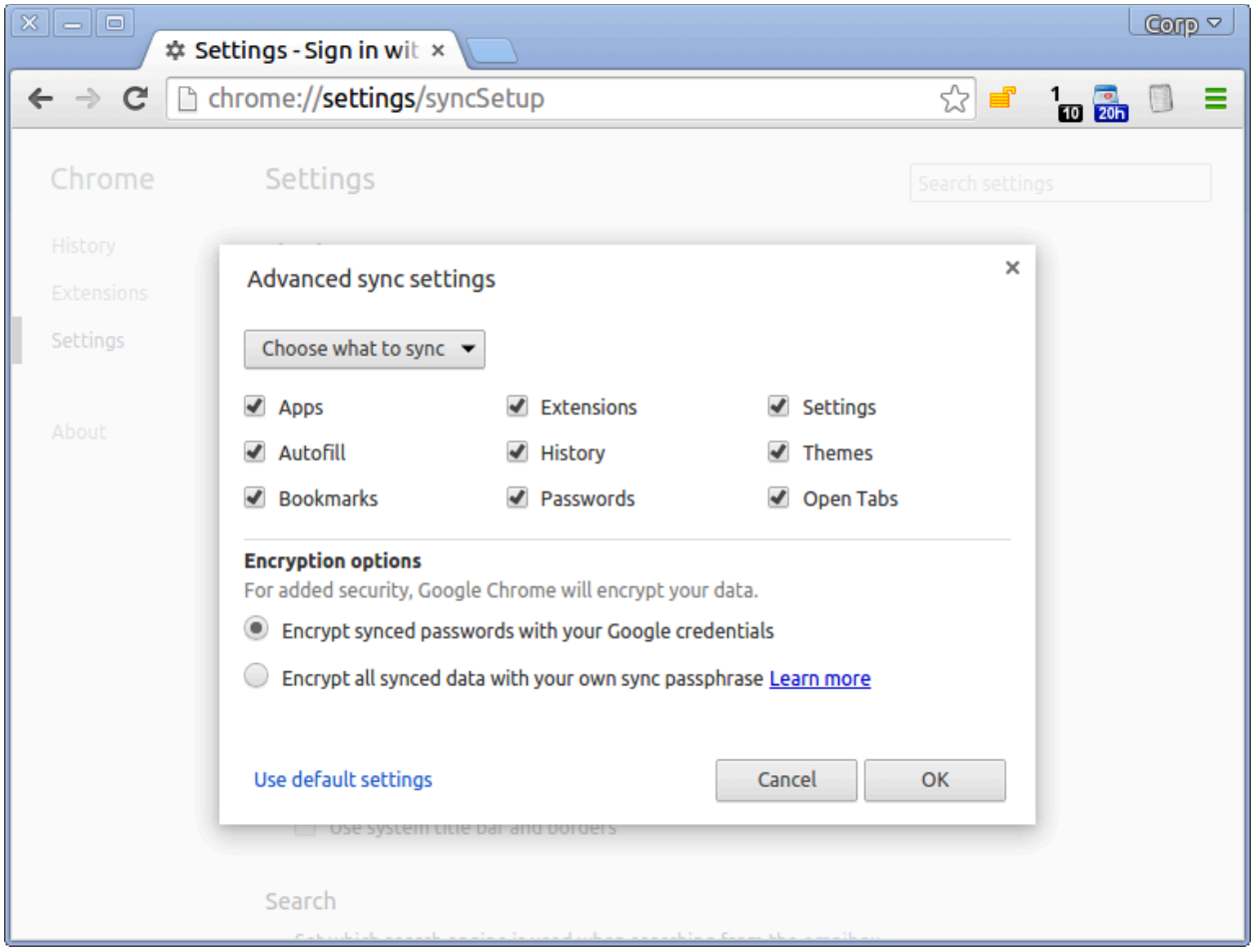
Video Zoom Extension

The interesting part about this is that it's written by the author of an options framework, as mentioned in crbug.com/25317.



Advanced sync settings




Settings - Sign in wit
Corp
chrome://settings/syncSetup
Chrome
Settings
Search settings
History
Extensions
Settings
About
Advanced sync settings
Choose what to sync
Apps
Extensions
Settings
Autofill
History
Themes
Bookmarks
Passwords
Open Tabs
Encryption options
For added security, Google Chrome will encrypt your data.
Encrypt synced passwords with your Google credentials
Encrypt all synced data with your own sync passphrase Learn more
Use default settings
Cancel
OK
Search