- **This is a template.** Please make a copy for your own API proposal.

chrome.yourApiNamespace - New Chrome API Proposal

Proposal Details

Proposal Date
✎ It's ok if this isn't perfectly accurate.

Who are the primary Eng and PM contacts for this API? (email addresses)
✎

What are the relevant bug(s)?
✎ crbug.com/…

Which team will be responsible for this API? (team email address)
✎ We ask that your team be an OWNER for this API implementation. It's yours, for better or for worse, for richer, for poorer...

Do you know other projects (internal or external) that are interested in this API?
✎ We like APIs that are going to be used rapidly after they are completed. If you know of anyone that will use it (or have an idea of how someone else could), let us know.

What's a reasonable Chrome milestone target for landing most of the code on Trunk?
✎ For example, M66. See https://www.chromium.org/developers/calendar for dates.

Will this API be public or private?
✎ We vastly prefer public APIs to private APIs (see also this section). If this is going to be a private API, why does it need to be?

Draft Manifest Changes
✎ Please mention new permissions and any other new manifest attributes.

Draft API spec (or changes)
✎ (If modifying an existing API, please link the file here and only draft the changes.) API specs can be written either as JSON or IDL. If writing a new spec, IDL is typically preferred because of increased readability. See existing API specs at either src/extensions/common/api or src/chrome/common/extensions/api.

Note that: (a) API calls that affect the browser process must be asynchronous (to prevent extension JavaScript from blocking on the browser UI), and (b) chrome.runtime.lastError is the standard way to signal Chrome errors and there's no reason to invent your own.

Overview

✎ Give a high-level description of the API you wish to add.

Use cases

✎ Explain why you need this API, and how it might be used by other apps/extensions.

Can you imagine any other similar use cases which your API might address?

1. ✎
2. ✎
3. ✎

Alternatives, Web Platform Considerations, and Stability

How would you implement your desired features if this API didn't exist?

✎ In some cases, good alternatives are a reason not to implement an API. Take care in answering this question, but additionally consider: can you just build it into Chrome (particularly for private APIs)? Is the use case really very limited and a workaround is easier (though this may be a reason for us to provide something better)?

Is this something that could/should be part of the web platform, or an existing chrome.* API?

✎ Keep in mind that Web APIs are preferable to Chrome APIs when reasonable. If you think your use case could be generally useful on the web send an email to blink-dev@chromium.org for their opinion. If not, we have a lot of existing Chrome APIs addressing a substantial body of use cases; look through our existing extension and app APIs to see if any are suitable.

Does this API expose any functionality to the web?

✎ If this exposes functionality to the web (for example, web-to-app messaging), then we'll make sure that a few engineers on the Blink team participate in reviewing this API. You will eventually need to email blink-dev@chromium.org about this one, too.

Do you expect this API to be fairly stable? How might it be extended or changed in the future?

✎ One of our core tenets is that we try to maintain backwards compatibility. Therefore, we prefer only expose APIs once we know they're not going to change (in an incompatible way). However, if it's not clear where the API needs to go it's reasonable (in fact, advisable) to only expose to dev channel, or even whitelist it to certain extensions/apps, but please don't let it just sit in that state.

If multiple apps/extensions used this API at the same time, could they conflict with each other? If so, how do you propose to mitigate this problem?
✎ We prefer APIs that multiple extensions can use gracefully. For example, any number of extensions can provide browser action buttons, and we don't allow extensions to fight over which ones are visible. In cases where such grace is not possible, we at least require understandable/consistent behavior. For example, extensions can override content settings, and these settings can conflict. We display when an extension is overriding a setting in the preferences UI and roll settings back to their previous values when an extension is uninstalled.

Security

How could this API be abused?
✎ Describe any concerns you have with exposing this API. Particular attention should be given to issues of security, privacy and performance.

Imagine you're Dr. Evil Extension Writer, list the three worst evil deeds you could commit with your API (if you've got good ones, feel free to add more):

4. ✎
5. ✎
6. ✎

What security UI or other mitigations do you propose to limit evilness made possible by this new API?
✎ We prefer to avoid warning messages in the extension install dialog whenever possible. Examples of particularly good approaches from past APIs:

- The chrome.debugger API shows an infobar across the top of a tab when a debugger is attached. If the infobar is closed, the debugger is detached. Thus, it's always obvious to a user when a debugger is attached, and there's no need for scary warnings during installation.
- The chrome.infobar and chrome.contextMenus APIs both display the extension's icon so that users cannot confuse the new UI for something provided by the system.

However, we do show install-time permission warnings in many cases. If your API should have one, describe it here.

Could a consumer of your API cause any permanent change to the user's system using your API that would not be reversed when that consumer is removed from the system?
✎ Another main tenet of the Chrome Web Platform is that apps/extensions should "leave no trace" when they are removed. If someone using your API could leave lasting changes, we need to know.

Other Considerations

List every UI surface belonging to or potentially affected by your API:
✎ Chrome's UI is simple and minimal and we try to keep it that way, it's very important to know which APIs can affect what areas. Furthermore with apps APIs - apps are browser agnostic, and any Chrome UI is discouraged or even impossible. Add mocks for every new UI surface introduced/affected by this API.

Actions taken with app/extension APIs should be obviously attributable to an app/extension. Will users be able to tell when this new API is being used? How? Can it be spoofed?
✎ For example the chrome.debugger API shows an infobar across the top of a tab when a debugger is attached. Familiar UI that can't be spoofed is ideal.

Does this API impose any requirements on the Chrome Web Store ?
✎ For example the URL handling API for apps requires that the app prove domain ownership of the URL that it wants to handle at the time the app is uploaded to the Chrome Web Store.

Does this API have any interaction with other Chrome APIs? Does it impose any restrictions on other APIs that can be used in conjunction?
✎ For example the chrome.pushMessaging API requires that the user be signed-in. Changes to the Identity API could impact the chrome.pushMessaging API. Another example is that an extension cannot have both a pageAction and a browserAction.

Open questions

✎ Note any unanswered questions that require further discussion.