WebExtension Experiments Documentation

Release 0.1

Add-ons team

Nov 30, 2017

Contents

WebExtensions are a cross-browser system for developing browser add-ons.

WebExtensions Experiments allow developers to write experimental WebExtensions APIs for Firefox. They can be used to prototype APIs for landing in Firefox, or for use on Nightly or Developer Edition.

If you simply want to request a WebExtensions API, please file a bug. These will be triaged and processed in a weekly public meeting.

Contents:

CHAPTER 1

Why?

WebExtensions Experiments provide a way for developers to tinker with new APIs for WebExtensions. They work by allowing WebExtensions APIs to be written in an another extension. They can be used to prototype APIs for landing in Firefox, or for use on Nightly or Developer Edition.

Note: If you simply want to request a WebExtensions API, please file a bug. These will be triaged and processed in a bi-weekly public meeting.

Note: If you'd like to land a WebExtensions API straight into Firefox, are familiar with building mozilla-central, working with Bugzilla, the try server, then please file a bug and follow the usual Firefox development process.

1.1 Overview

Terminology:

experiment A bootstrapped add-on that contains code to expose a WebExtensions API.

extension A WebExtension add-on that uses the experiment add-on as a dependency.

Note: The *experiment* and the *extension* are both add-ons. The *experiment* is a legacy style add-on, while the *extension* is a WebExtension.

Experiments allow you to:

- Test and experiment with an API without having to build Firefox at all.
- Write and distribute the API to a set of users without them having to build Firefox.
- Then commit to (or get help committing to) mozilla-central.

Note: We will not add all experiments to Firefox. The goal is to judge each Experiment on its own merit and value.

Some key points:

- The API is implemented in the experiment add-on.
- The API is available in the browser namespace - not in chrome.
- Breaking changes could occur in the experiment, but are discouraged.
- WebExtension add-ons using the API depend on the experiment add-on.
- WebExtension add-ons using the API declare its use in the manifest permissions.

1.1.1 How do they work?

Experiments are a bootstrapped add-on. They have a special type *256* that tells Firefox that they are an experiment. Firefox loads the *schema.json* into Firefox and then the APIs become available to WebExtensions.

1.1.2 Where do they work?

Currently experiments can only be loaded in Firefox Nightly and Firefox Developer Edition.

Please see the *Policy* page.

CHAPTER 2

Existing Experiments

This is a list of the experiments primarily from this organisation on github. Since anyone can run an experiment this should not be seen as a definitive list.

2.1 Login

Access to the *nsILoginInfo* object.

status **experimenting**

source https://github.com/web-ext-experiments/logins

AMO https://addons.mozilla.org/en-US/firefox/addon/experimental-logins-api/

documentation https://github.com/web-ext-experiments/logins/blob/master/api.md

addon-id and downloads logins@experiments.addons.mozilla.org

Add-ons using it:

- https://github.com/andymckay/password-exporter

2.2 Hello world

A simple example of Hello World which logs to the console.

status **example**

source https://github.com/aswan/webext-experiment-hello/tree/master/experiment

documentation https://github.com/aswan/webext-experiment-hello

addon-id and downloads simple@experiments.addons.mozilla.org

Add-ons using it:

- https://github.com/aswan/webext-experiment-hello/tree/master/extension

CHAPTER 3

Roadmap

3.1 Trello

This is a list of the experiments from the WebExtensions roadmap on Trello. In cases of discrepancy, bugzilla > trello > this page are the definitive sources.

3.1.1 Legend

- Done: This experiment is done and merged into mozilla-central.
- Experimenting: The experiment exists and is listed here, but isn't ready into mozilla-central yet.
- In development: Someone is working on it.
- Approved: Sounds like a good idea, someone needs to work on it.
- Thinking about: Still thinking if this a good idea.

CHAPTER 4

How to write an Experiment

An experiment comprises at least three files:

- *api.js*: Your code to implement the API.
- *schema.json*: Defines the API - types, methods, etc.
- *install.rdf*: Install manifest.

The namespace is the value used in the API such as "login" or "media". This is what developers will use in the API, such as "browser.login" or "browser.media". The namespace needs to be consistent in all places of the extension:

- *api.js*: ExtensionAPI.getAPI() must return an object with a single top-level property: "namespace".
- *schema.json*: the permissions must be *"namespace"*.
- *install.rdf*: the id must be *"namespace@experiments.addons.mozilla.org"*.

You can see an example of all this in the boilerplate example where the namespace is "boilerplate".

4.1 Using the boilerplate

The boilerplate example provides a simple example that you can download and use in whatever way you see fit.

To get the boilerplate example on Linux or OS X, run the following command:

```
curl -L https://github.com/web-ext-experiments/boilerplate-experiment/archive/master.
↪tar.gz | tar zxf -
```

You will then be able to change the example to meet your needs.

4.2 Designing your API

You are free to do whatever you'd like to do in an experiment, but if you eventual goal is to uplift it into Firefox then you should probably read the *Uplifting* and *New API Guidelines* documentation.

4.3 Installing the experiment

- Note: Due to the current Extension Signing policy, you will need to use the Nightly, Developer Edition or Unbranded builds and set the "xpinstall.signatures.required" preference to "false" to allow install unsigned add-ons.

Your experiment can be installed like any other add-on, for example:

1. You can zip an *.xpi*, and then install it from *about:addons* > *Install Add-on From File*.

```
zip experiment.xpi api.js install.rdf schema.json
```

2. You also can load the install.rdf temporarily from *about:debugging* > *Load Temporary Add-on*.

4.4 Using the experiment

Once you have experiment installed, you can then create a WebExtension using the API provided in the experiment. The only requirement is to add into the manifest of the WebExtension the permission for the experiment.

For example from logins:

```
"permissions": ["experiments.logins"]
```

This gives you permission to use the API. But the API might require more permissions, for example to actually use the *logins* API for all urls, you'll need to do the following:

```
"permissions": ["experiments.logins", "logins", "<all_urls>"]
```

Please see the documentation for each experiment to find what permissions you need.

CHAPTER 5

Uplifting

Once you've got an experiment the hope is that it gets merged into Firefox and mozilla-central. But before that happens there's a few things that should happen.

5.1 Adding to github

Note: This is optional.

5.2 Tell people

The first step is to file an issue against this repository. In that issue please outline:

- where your experiment is located
- a quick overview of what it does
- any bugzilla bugs that it might address
- if you'd like to move your repository over to this organisation and we can create a repository for you

We'll then include these in our bi-weekly community triage meeting.

There's a few emails and IRC channels that would also like to know about your experiment.

5.3 Updating this documentation

It would be great to add it to this documentation too. If you would like to, please send a pull request to this repository.

5.4 Landing in Firefox

Note: This is optional.

For it to land in mozilla-central a few things are going to have to happen.

- It should follow the code quality guidelines.
- There should be mochi and xpcshell tests at 100% coverage of the API.
- There should be some documentation on the API.
- There should be a review (even if its just cursory) by the following: the security team, the privacy team, a code review by the engineering team and the UX team (where relevant).
- If the API requires a permission, ensure to land the corresponding permissions string.

For more information on what is likely to be accepted, please check out the *New API Guidelines* API Guidelines.

The best way to do this is by filing a bug in Bugzilla and we can start to work through the steps. File a bug under Toolkit > WebExtensions: Experiments and that process can be started.

5.5 What if it doesn't land in Firefox?

It can still be used by Nightly and Developer Edition.

CHAPTER 6

New API Guidelines

There are some basic principles around WebExtensions APIs as they currently stand. Any API that wants to get merged into Firefox would need to meet these principles.

- **Security**: the API should not expose an unacceptable risk to the end user.
- **Privacy**: similar to above.
- **Performance**: APIs should be async by default and focus on not causing jank, hangs or any sort of bad performance in Firefox.
- **Multi-process**: all APIs need to be multi-process aware.
- **Useful**: APIs that land in Firefox should be useful since all APIs have a maintenance burden.
- **High level**: WebExtensions provide a known public API layer on top of Mozilla code, allowing the underlying code to change quicker and easier.
- **Alternatives**: Do alternatives exist for example, WebAPIs?

Compared to legacy Firefox add-ons there is a significant sandbox around WebExtension APIs that is very restrictive. Access to priviledged APIs within Firefox, preferences, arbitrary File System access, socket access and so on are restricted (at the time of writing this documentation). APIs that break out of this sandbox will need very clear justification for why.

6.1 General principles

As we've been triaging bugs, we've come up with some general principles and guidelines. These are liable to change and adapt as we go along.

6.1.1 Users over Developers

One example was the request to prevent users from dragging browserAction buttons into the overflow menu, tab menu or customize menu. This bug was denied because its the users choice where they place their buttons, *not* the developers.

6.1.2 Users understanding UI changes

If a UI change is going to occur, it should occur on install of the extension or in a user action. This makes it clearer to the user where the change came from.

Where possible any UI that is added should have some clear trace back to the extension that installed it, maybe through icons or text.

6.1.3 Cleaning up on uninstall

Firefox will do its best to clean up on uninstall, for example flipping preferences back and reverting UI changes. When we develop an API its important to think about how this would work when the extension is uninstalled. It should not be a burden developers have to carry.

6.1.4 Complexity and maintenance

For every feature we have to make an rough evaluation of the cost to implement and the benefit its hoped it will bring. But in there is also the cost of maintenance for that feature. Firefox has been around since 2004 and the maintenance burden of the code in extensions cannot be underestimated.

6.1.5 Anti-patterns

Here are the things that are red flags for any API:

- "We'll just make sure we manually review all extensions using this." This is a common request for APIs that have large security or privacy issues. It pushes the burden over to Mozilla and volunteers to review the extension. This is costly, time consuming and disliked by developers. With the changes to post-review by Mozilla, this is no longer an option.
- Changing the UI dynamically at runtime is normally a big concern.
- Anything that is outside the area of making Firefox interact with web pages (please see the Vision document)

6.2 Permissions

If you add in a new API, you will need to consider a permission for that API. Permissions have multiple purposes.

1. They allow quick static analyis of the extension to see what APIs they use. At Mozilla we use this to calculate API popularity and contact authors in the case of API issues.
2. They can prompt the user on install of the extension to let them know what it does. Having a permission doesn't mean it has to prompt.
3. They can be used optionally to give permission, but not all APIs should be used optionally.

Generally we prompt for a permission if it:

- Means that the extension has access to user data from web pages that isn't immediately obvious to user. For example: a content script on <all_urls> will have access to anything on any website the user visits.
- Means that the extension will reach outside of the sandbox. For example: nativeMessaging allows Firefox to talk to external programs on the users computer.
- Any other security or privacy issues that might need consideration.

An example of a permission that does not prompt is: idle.

6.3 APIs and the W3C

We are a member of the community. However expecting other browsers to standardise shouldn't slow down or prevent development. Any changes should eventually be brought to the community group, in the hope that other browser can support them. We'll mark all APIs appropriately on MDN to show their status for the standards track.

6.4 What have we allowed

When a feature request comes in we try to mark it with [design-decision-needed] in Bugzilla. Not all bugs get this designation, just ones that we think we need to talk about. From that meeting we try to come out with one of the following designations:

- [design-decision-needed]: it's waiting in the queue for the meeting.
- [design-decision-deffered]: we really don't know at this point and we'll hopefully get to it later.
- [design-decision-approved]: sounds good, if someone landed a patch for this we'd accept it. The details might still need to be worked out though.
- [design-decision-denied]: we wouldn't accept this patch if you did work on it.

The point of this process is to prevent people from putting hard work into something only to have someone say no. To see all the bugs in these categories, checkout the following queries:

- decision approved
- decision denied

These might give you an idea of what might make it. Please note that decisions are not final and have been reversed in the past, nothing is final or perfect.

6.5 Triaging APIs

If you file a bug asking for an API it will go through the bug triage process. Since this is useful to how APIs land, here's what currently happens in bug triages:

6.5.1 Weekly new bug triage

We look at all new bugs in an attempt to spot serious bugs, regressions or other issues. We try to give each bug a priority. The point here is to do an initial triage and catch critical things. We also label bugs that might be good for contributors or need thinking about. The latter are marked *[design-decision-needed]*, but its important to point out that straightforward change or obvious bugs just go through.

6.5.2 Bi-weekly community meeting

We look at a number of bugs marked with *[design-decision-needed]* every other week, currently we are doing 6 per meeting, which averages us at 5 minutes per bug. Some take longer. The goal here is to see the use case, understand what the bug is for and if it should proceed. If we are still unsure then it will get kicked "up" to the Advisory Group for some more help and insight.

6.5.3 Good first bug meeting

If a bug gets marked as *[good-first-bug]* then we make sure the bug has a mentor, has a decent description and make sense. We hope that contributors will use this to get into Firefox development.

If you'd like to come to some of these triages, check out the calendar: https://wiki.mozilla.org/Add-ons#Meetings

CHAPTER 7

Policy

Note: This policy is of Oct 2016 and should be seen as a **first draft**.

Experiments require Firefox 50 or greater to work.

Anyone can write a WebExtension Experiment, there is no need to ask for permission from anyone to do so.

Currently Experiments can only be run on Firefox Nightly or Firefox Developer Edition. This is to prevent APIs that are generally seen to be experimental being run on release versions of Firefox.

CHAPTER 8

Schema

What the schema file is and how it works.

CHAPTER 9

FAQ

WebExtensions Experiements have many similarities to traditional add-ons, therefore a lot of general documentation is already available on developer.mozilla.org. The following questions and answers are specific to WebExtensions experiments and should help you with common issues.

9.1 How to produce log messages from an experiment api.js file?

On Firefox versions >= 55, the `console` API object is already available in the sandboxed environment where the `api.js` file is executed, on Firefox versions where it wasn't yet available you can define it using the `Console.jsm` module:

```
const {ConsoleAPI} = Cu.import("resource://gre/modules/Console.jsm", {});

const console = new ConsoleAPI({
  prefix: "webext-experiment-MYAPINAME",
});
```

9.2 How do I get an URI reference to the files in my experiment?

You may need an URI referencing files within your extension, for example to load a frame script when your API methods are being triggered. If you have a file named `browserScript.js` in the root of your add-on, you can reference it using the URI `resource://extension-NAMESPACE-api/browserScript.js`, where `NAMESPACE` is the namespace defined in your `schema.json` file.

9.3 Can I include a chrome.manifest in an experiment?

The experiments do not support the 'chrome.manifest', and it will be ignored if included in the experiments add-on source directory. If you need an URI which points to assets packaged within the experiment (e.g. an image, an

additional .jsm file, a frame script, etc.), please read the *question about URI references*.

If you have any more questions you can't find an answer to please get in touch.

This documentation lives on github, pull requests and issues are welcome.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search