

File System Access API

About me



Hi, I'm Vlad, software engineer.



Lineate Team Member [↗](#)

Occupied software engineer position

2

Fork me on GitHub

Plan

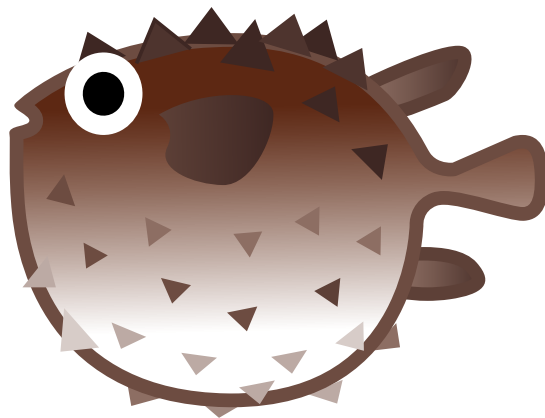
Careful, spoilers



Plan

1. Project FUGU;
2. Web platform and tests
3. File System Access API
 1. What is that
 2. Why is that
 3. Pros and Cons
 4. Security
 5. Demos
 6. Questions

Project FUGU



The capabilities project is a cross-company effort with the objective of making it possible for web apps to do anything iOS/Android/desktop apps can, by exposing the capabilities of these platforms to the web platform, while maintaining user security, privacy, trust, and other core tenets of the web.

You can see the full list of new and potential capabilities and the stage each proposal is in on the

[FUGU🐡 API tracker](#)



8

Fugu API Tracker

STABLE



Chromium 98

Stable 3 days ago
(Feb 1, 2022)

BETA



Chromium 99

Stable in 25 days
(Mar 1, 2022)

DEV



Chromium 100

Stable in 53 days
(Mar 29, 2022)

You can even suggest your own feature of capability 😎

If you are interested you can join conversations there

Mailing list fugu-dev@chromium.org

Slack Channel: #fugu on [@chromium](#)

Alright, we're done with ads, let's finally get back to the point!



Let's take a look at what FUGU page says about File System Access AP






12

→ ↻ 🏠 🔒 fugu-tracker.web.app/#shipped

📄 📁 ⭐ 🐱

Shipped

File System Access



This API enables developers to build powerful apps that interact with other (non-Web) apps on the user's device via the device's file system. After a user grants a web app access, this API allows the app to read or save changes directly to files and folders selected by the user. Beyond reading and writing files, this API provides the ability to open a directory and enumerate its contents, as well as store file and directory handles in IndexedDB to later regain access to the same content.

Consensus & Standardization

Browser	Status
Edge	No Signal
Firefox	No Signal
Other	
Safari	No Signal
Developers	Positive

Implementation Information

Tracking Bug	853326 📄
Specification	https://wicg.github.io 📄
Chrome Status	Details 📄
Stars	103

Explainers

- <https://github.com>

Demos

- <https://github.com>
- <https://googlechromelabs.github.io>

Documentation

- <https://web.dev>
- <https://web.dev>

Bug tracker for Blink browsers of FileSystem Access

Chromium status

Let's take a look at what FUGU page says about File System Access AP

13

→ ↻ 🏠 🔒 fugu-tracker.web.app/#shipped

Shipped

File System Access

M86

🐧 🪟 🌐 🍏

This API enables developers to build powerful apps that interact with other (non-Web) apps on the user's device via the device's file system. After a user grants a web app access, this API allows the app to read or save changes directly to files and folders selected by the user. Beyond reading and writing files, this API provides the ability to open a directory and enumerate its contents, as well as store file and directory handles in IndexedDB to later regain access to the same content.

Consensus & Standardization	
Browser	Status
Edge	No Signal
Firefox	No Signal
Other	
Safari	No Signal
Developers	Positive

Implementation Information	
Tracking Bug	853326 📄
Specification	https://wicg.github.io 📄
Chrome Status	Details 📄
Stars	103

Explainers

- <https://github.com>

Demos

- <https://github.com>
- <https://googlechromelabs.github.io>

Documentation

- <https://web.dev>
- <https://web.dev>

Bug tracker for Blink browsers of FileSystem Access

Chromium status

By the way, it was first shipped in Chrome 91



Let's take a look at web platform experimental features tests failures

Browser-specific failures are the number of WPT tests which fail in exactly one browser. This graph shows the BSF scores for Chrome, Firefox and Safari.

Channel

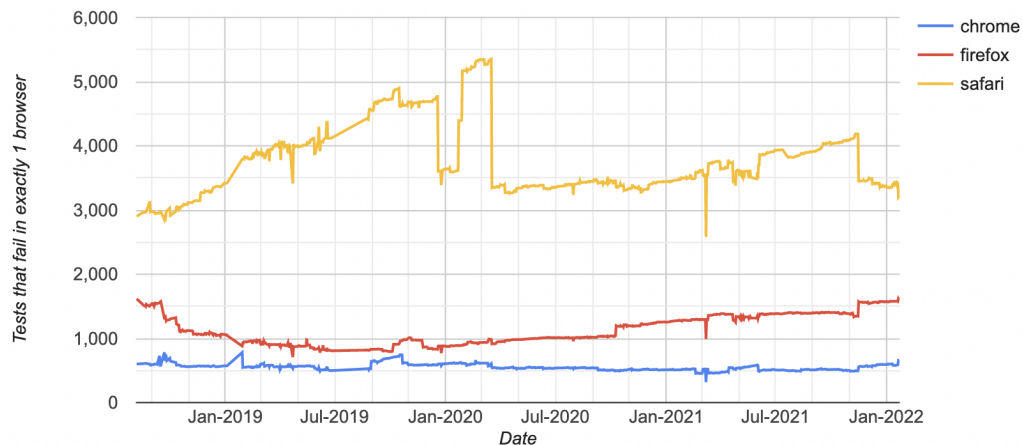
Experimental

Stable

Last updated WPT revision

da87c9bcf5

Click + drag on graph to zoom, right click to un-zoom



What do we see?

Browser-specific failures are the number of WPT tests which fail in exactly one browser. This graph shows the BSF scores for Chrome, Firefox and Safari.

Channel

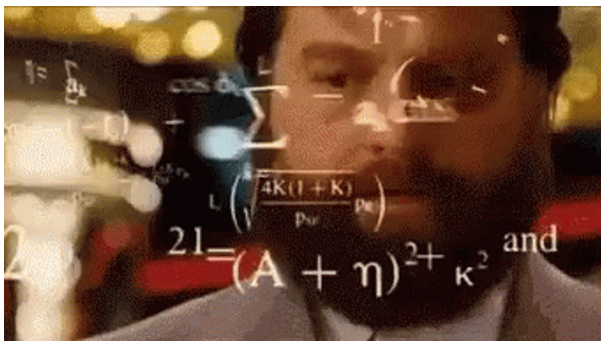
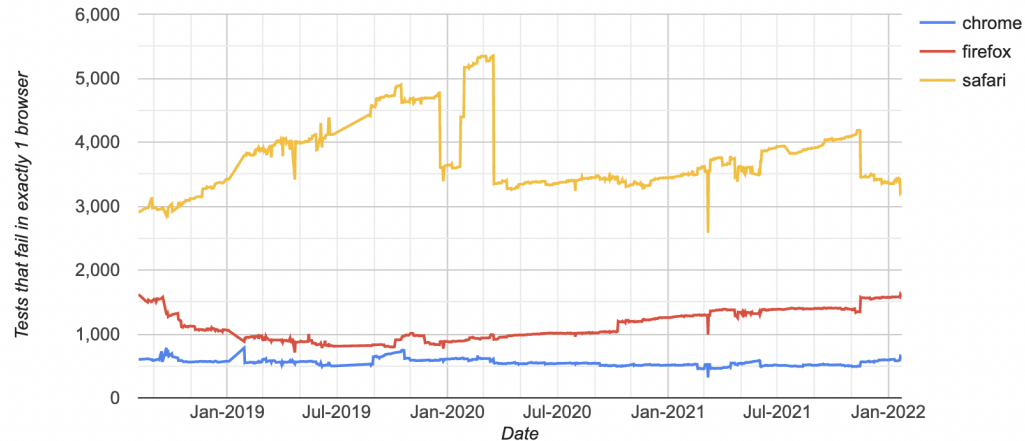
Experimental

Stable

Last updated WPT revision

da87c9bcf5

Click + drag on graph to zoom, right click to un-zoom



Right! Nothing, my curious minds!



Right! Nothing, my curious minds!



Because there are no tests on web platform for File System API just yet

But what is File System Access API? What does it mean for those all who uses web apps?

But what is File System Access API? What does it mean for those all who uses web apps?

Well, let's figure it out

File System Access API is obviously is the application programming interface.



More formally, how the first implementer states:

“ This API enables developers to build powerful apps that interact with other (non-Web) apps on the user’s device via the device’s file system. (c) Chrome (taken from [here](#)) ”

Specifications

Specification	Status	Comment
File System Access API 	Working Draft	Initial definition.

<https://wicg.github.io/file-system-access/>

Specifications

Specification	Status	Comment
File System Access API 	Working Draft	Initial definition.

<https://wicg.github.io/file-system-access/>

This particular spec is developed by W3C (The World Wide Web Consortium)



What is the reason for that feature?

In native applications, there are common file access patterns.

In native applications, there are common file access patterns. For instance (all examples are high-level):

1. For Android (via Kotlin or Java):

1. [File API](#)
2. [Open files using stream](#)
3. etc

2. For iOS:

1. [File Manager](#)

3. For desktop:

1. Windows APIs
2. Linux Filesystems API

But are there any for browsers?

But are there any for browsers? Spoiler: topic of report says about it 🧐

Let's consider cases when you may need to interact with FS (File System)

According to opinion of chrome team, the reason to interact with file system is to interact with native applications.

Here it is

“ The main overarching goal here is to increase interoperability of web applications with native applications, specifically where it comes to being able to operate on the native file system. ”

Cases when you may need it

- [single file editor](#);

Cases when you may need it

- [single file editor;](#)
- [multi-file Editor](#)

Cases when you may need it

- single file editor;
- multi-file Editor;
- file libraries;

How to use?

[Show how it works!](#)

```
1  const pickerOpts = {
2    types: [
3      {
4        description: "Images",
5        accept: {
6          "image/*": [".png", ".gif", ".jpeg", ".jpg"],
7        },
8      },
9    ],
10   excludeAcceptAllOption: true,
11   multiple: false,
12 };
13
14 async function getTheFile() {
15   // open file picker
16   [fileHandle] = await window.showOpenFilePicker(pickerOpts);
17
18   // get file contents
19   const fileData = await fileHandle.getFile();
20 }
```

Legacy for file picking

No file chosen

Are there any
real-life
examples of usage of that brand new API?

Of course there are 🥰🥰🥰

Check them!

It works poorly in iframe that I use for demos, but here is the [link](#)

It works poorly in iframe that I use for demos, but here is the [link](#)

Unfortunately, I am not invited to beta-testing of Photo shop for web,
but this states that there is the Adobe PS for Web

OK





But what is under the hood?

A bit of terminology



An

entry

is either a file entry or a directory entry.

An

entry

is either a file entry or a directory entry.

i.e. something that represents file or directory in terms of FSA API

The FSA API builds on top of

File API

So let's take a look at terms for File API

A Blob object refers to a byte sequence, and has a size attribute which is the total number of bytes in the byte sequence, and a type attribute, which is an ASCII-encoded string in lower case representing the media type of the byte sequence.

Term defined by File API spec made by W3C

A Blob blob has an associated get stream algorithm, but we'll stick to that fact only otherwise it is for a long time

We better take a look at how FSA API and Blobs are related

Interface of File Entry has method called

createWritable

Interface of File Entry has method called

createWritable

it returns `FileSystemWritableFileStream` object that is in fact

[WritableStream \(according to WHATWG Streams Spec\)](#) object

Interface of File Entry has method called

`createWritable`

it returns `FileSystemWritableFileStream` object that is in fact

[WritableStream \(according to WHATWG Streams Spec\)](#) object

and has method called `write()` that accepts object of properties where Blob data can be found

OK





But what about its issues? Looks a bit flawless, doesn't it?

There are so many of them that I would be able to create a new little report about them.

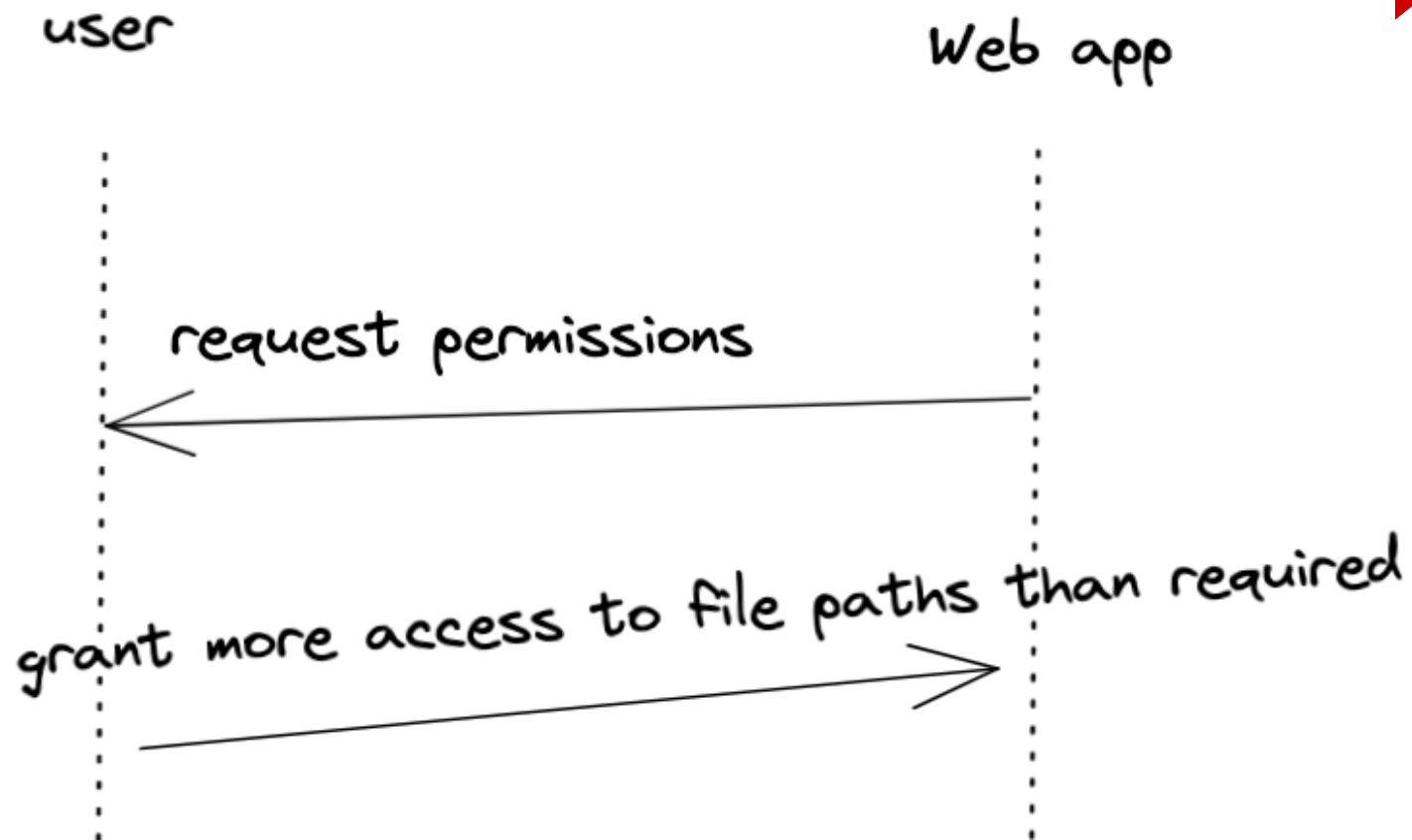
► Note: I'd like to talk about problems that were issued by spec authors! ►

Security Risks

Security Risks

1. Electron does not update regularly and also is about to not to effectively defend XSS
2. Small utilities, such as archivers and image/PDF resizing/cropping, are often native applications that run with the user's full privileges. Some of these applications deliver malware.
- 3.

Worth mentioning the following attack



user executes file the web app saved to a disk



user

Web app

request permissions

grant more access to file paths than required

Malware being written to file.

► Details

Ransomware attacks

► Details

Fillin up a user's disk

► Details

ISSUE 1 We should consider having further normative restrictions on file names that will never be allowed using this API, rather than leaving it entirely up to underlying file systems.

ISSUE 2 TODO: Explain better how entries map to files on disk (multiple entries can map to the same file or directory on disk but an entry doesn't have to map to any file on disk).

ISSUE 4 Currently `FileSystemPermissionMode` can only be `"read"` or `"readwrite"`. In the future we might want to add a `"write"` mode as well to support write-only handles.
<<https://github.com/wicg/file-system-access/issues/119>>

ISSUE 6 There has been some discussion around and desire for a "inPlace" mode for createWritable (where changes will be written to the actual underlying file as they are written to the writer, for example to support in-place modification of large files or things like databases). This is not currently implemented in Chrome. Implementing this is currently blocked on figuring out how to combine the desire to run malware checks with the desire to let websites make fast in-place modifications to existing large files. [<https://github.com/wicg/file-system-access/issues/67>](https://github.com/wicg/file-system-access/issues/67)

► Details