File System

Austin Sullivan - asully@google.com - GitHub: @a-sully

TPAC 2022

Specifications: whatwg/fs + WICG/file-system-access

Agenda

- Introduction and history
- Current state of things
- Where do we go from here?
- Discussion!

History

- Work in this space has happened as far back as 2012
- Another abandoned attempt by Mozilla in 2015
- File and Directory Entries API used for drag & drop
  - `webkitRequestFileSystem()`
- Early discussions of what would become the File System Access API at TPAC in 2018 (led by mek@google)

> The File System Access API enables developers to build powerful web apps that interact with files on the user's local device.

The File System Access API: simplifying access to local files
https://web.dev/file-system-access/

The File System Access API (WICG/file-system-access)

- File and directory pickers
- Integrated with Drag & Drop
- Can store handles in IndexedDB
- Write files using a modern streams-based `createWritable()` API
- The File Handling API grants File System Access handles

Use cases:

- Web IDEs and text editors
- Photo and video editors
- . . .

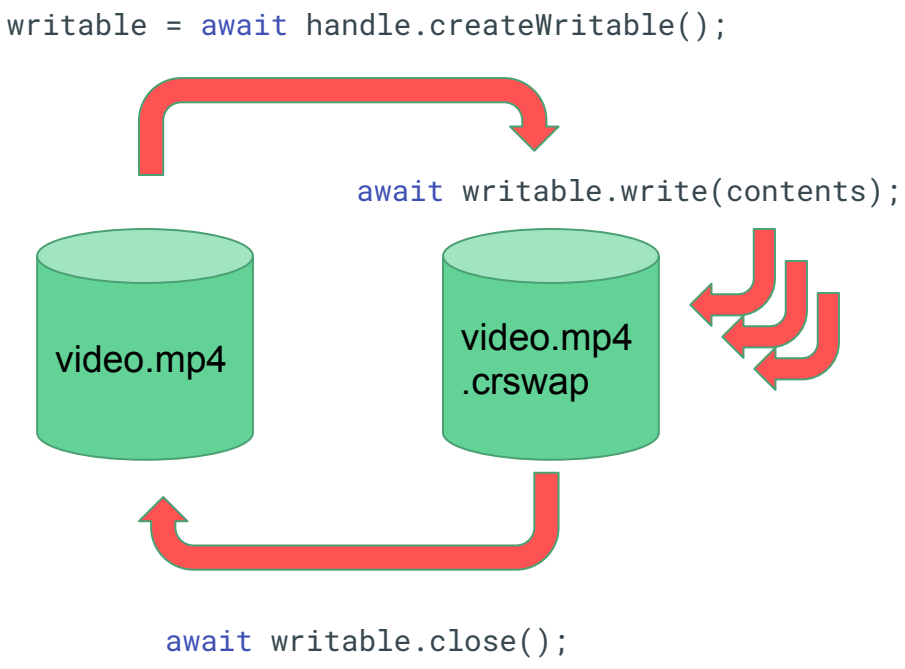
```
// User file system
handle = await showOpenFilePicker();
handle = await showSaveFilePicker();
handle = await showDirectoryPicker();

// Origin private file system
handle = await navigator.storage.getDirectory();

// Writing a file
writable = await handle.createWritable();
await writable.write(contents); // To swap file
await writable.close();

// Permissions
await handle.queryPermission(options);
await handle.requestPermission(options);

...
```

Feedback from power users

Requiring Safe Browsing checks on file edits has significant implications

Cannot support in-place writes

Significant performance challenges, especially with respect to large files and sequential writes



What about the Origin Private File System? (OPFS)

- Shipped as part of File System Access API
- Storage endpoint private to the origin of the page
- Contents not easily user accessible
- Safe Browsing checks not needed

```
handle = await navigator.storage.getDirectory();
```

Meanwhile, elsewhere in Chrome...

> The Storage Foundation API is a new fast and unopinionated storage API that [allows developers to] "bring their own storage" to the web, reducing the feature gap between web and platform-specific code.

Use cases:

- File I/O for C(++) applications compiled to WASM
- “BYODB”
- Editing large files
- Swap files

High performance storage for your app: the Storage Foundation API
https://developer.chrome.com/docs/web-platform/storage-foundation/

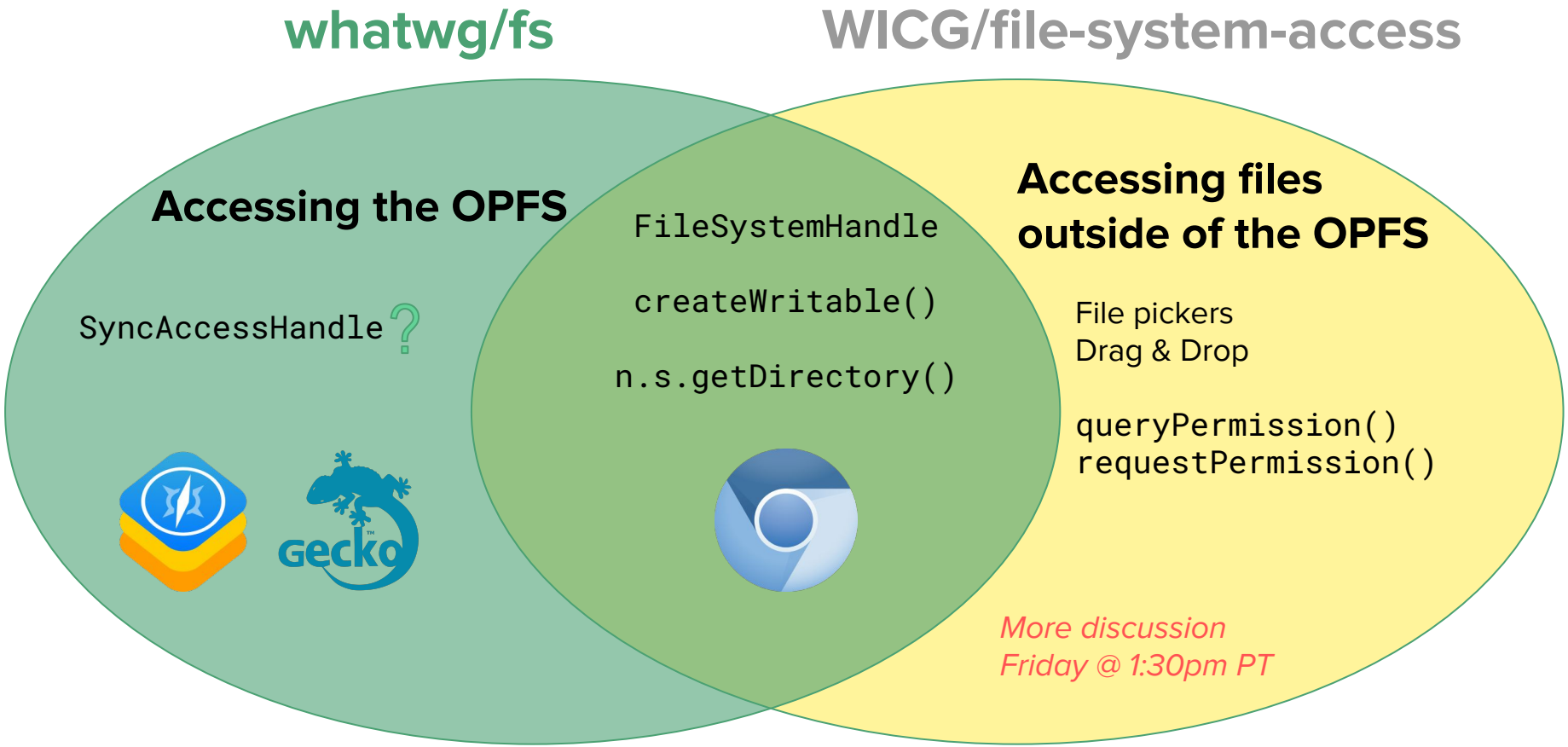
Merging Storage Foundation into File System Access (doc)

New idea: Sync Access Handles for the OPFS

- Ultra-fast file primitive
- Synchronous, POSIX-like file API
- Only available from dedicated workers
- Only available for files within the OPFS
- Cross-browser support 🎉

```
accessHandle = await fileHandle.createSyncAccessHandle();
bytesWritten = accessHandle.write(buffer);
readBytes = accessHandle.read(buffer, { at: 1 });
accessHandle.close();
```

One spec? Why not two?!



What we're doing now

- Responding to initial developer feedback
  - #7: Making SyncAccessHandles fully sync
  - Resolving inconsistencies in the spec (#21 still pending merge)
- Rounding out the API surface
  - #9: remove()
  - #10: move()
  - #46: getUniqueId()
- Expanding the reach of the API
  - Bringing the OPFS to Android
  - Working with the SQLite team to make SQLite-over-WASM-over-OPFS a reality

Where do we go from here?

- #34: Supporting multiple readers
  - Supporting multiple writers?
- #18: Explicitly define locking mechanisms
  - Relaxing the exclusive file locking requirement
  - Byte-range file locking?
- #41: Interest in an async alternative to SyncAccessHandles?
  - Promise integration for WASM coming down the pipe (eventually)
- Anything else we can fix while usage is low and concentrated?
  - #4: renaming `resolve()` to something more sensible

Where do we go from here?

- Other use cases we’re not aware of?
- Further gaps to address?