IOUtils Project Plan

A new ChromeOnly API for File I/O.

Author: Keefer Rourke
Last updated: 2020, Aug 27

Related docs

- Port OS.File to C++

Related bugs

Bug 1529336:

- OS.File loads scripts in a way that means they stick around in memory as strings

Bug 975702:

- [OS.File] Port OS.File to C++

Bug 1231711:

- [meta] Port OS.File to C++

Bug 986145:

- [OS.File] Stop using OS.File during startup

Abstract

OS.File (osfile.jsm) is a filesystem API for our frontend JavaScript code in Firefox. For several years, we have wished to port the OS.File API to C++ for performance gains (reduced memory usage, cache-contention, and disk access).

OS.File is implemented in both JavaScript and C++, but the main API for interacting with it is written in JavaScript. As a consequence of this, each process must load its own copy of the API, as it cannot live in shared memory. With the Fission project (site isolation for SPECTRE et al.) the number of Firefox processes will increase drastically, so this API must be rewritten.

I propose a new WebIDL API, dubbed “IOUtils” (named after ChromeUtils, DebuggerUtils, InspectorUtils, etc.), to be the replacement for OS.File.

This is a working document to address the issue of defining and implementing the new IOUtils API in C++, as well as the eventual replacement of OS.File. Goals (and non-goals) of the project are outlined below with a project plan, progress on implementations, and any additional concerns and questions.

Goals

- Create an ergonomic, unified, and performant API for File I/O, instrumented from JavaScript
- Feature parity with OS.File, except for broken features which have unaddressed bugs
- Use clear, simple method signatures for common usage patterns
- Fully describe the API in Web IDL to freely allow for the implementation to change
- Use this opportunity to simplify and/or improve the API used for file I/O
    - E.g. Attempt to hide platform-specific differences between win32/unix systems

Non-Goals

- Perform a method-by-method, class-by-class port; the shape of this new API may be different than OS.File's API for technical or preferential reasons
- To surprise; the new API should be familiar to the caller, following modern but established conventions in the mozilla-central code-base.
- Replace other I/O APIs (e.g. nsIFile & FileUtils) which may be present in mozilla-central
    - Corollary: We also do not want to introduce Yet Another File API, which will be half-adopted. This API must be stable, and sufficient to cover OS.File use-cases.
- Work outside the context of the Firefox desktop front-end; this API is intended to be exposed only to privileged JavaScript (ChromeOnly)

IOUtils API

This revised File I/O API takes inspiration from the existing OS.File API, and well-known I/O libraries such as Golang's os and io/ioutil packages, Apache Commons IOUtils, among others.

This API is privileged, and it should be clear that it is *not to be confused* with the unprivileged DOM File API.

Differences from OS.File

At no point should a file descriptor/handle be exposed to the caller. This API operates only on path ~~and file~~ objects. Platform-specific file descriptors may have been exposed by instances of OS.File through the `fd` attribute. This will no longer be possible.

The Date type does not exist in WebIDL, thus IOUtils will have no methods that support changing file access/modification times with specified values.

Error reporting

When an OS.File method runs into an error, it will throw/reject with a custom OS.File.Error object with custom attributes that can be checked for different common error cases.

IOUtils has similar behaviour, but instead it throws/rejects with a DOMException. DOMExceptions have a name and a message. The following DOMExceptions are thrown depending on the failure:

Exception Name	**Reason for exception**
`NotFoundError`	A file at the specified path could not be found on disk.
`NotAllowedError`	Access to a file at the specified path was denied by the operating system.
`NotReadableError`	A file at the specified path could not be read for some reason. It may have been too big to read, or it was corrupt, or some other reason. The exception message should have more details.
`ReadOnlyError`	A file at the specified path is read only and could not be modified.
`NoModificationAllowedError`	A file already exists at the specified path and could not be overwritten according to the specified options. The exception message should have more details.
`OperationError`	Something went wrong during the I/O operation. E.g. failed to allocate a buffer. The exception message should have more details.
`UnknownError`	An error happened that is unknown to the IOUtils implementation. The nsresult error code should be included in the exception message to assist with debugging and improving the IOUtils internal error handling.

Ownership and Review

This API will be ChromeOnly, and thus will be placed under the following source directories:

1. Web IDL files will be placed under dom/chrome-webidl
2. Implementations will be placed under dom/system

Therefore, this code will be owned by the Document Object Model Module, and reviews will be requested from peers of the DOM.

Specification

This privileged API will be exposed on both Windows and Workers in the Firefox front-end.

A patch with the first draft of the specification can be found here.

The current API is defined here. Example usages can be found in tests here and here.

Implementation

IOUtils will be implemented as a set of abstractions in C++, on top of the Netscape Portable Runtime (NSPR) to support a platform-neutral API and implementation.

Static methods

OS.File method	IOUtils equivalent	Implementation status
read	read	✅ (patch)
	readUTF8	✅ (patch)
writeAtomic	writeAtomic	✅ (patch) (follow-up)
	writeAtomicUTF8	✅ (patch)
move	move	✅ (patch)
rename		
remove	remove	✅ (patch)
removeDir	remove({recursive: true})	
removeEmptyDir		
makeDir	makeDirectory	✅ (patch)
stat	stat	✅ (patch)
setDates	touch	✅ (patch)

copy	copy	✅ (patch)
new DirectoryIterator(path)	getChildren	✅ (patch)
setPermissions	None	❌ Will not implement
unixSymlink	None	❌ Will not implement
getCurrentDirectory	None	❌ Will not implement
setCurrentDirectory	None	❌ Will not implement
open	None	❌ Will not implement
openUnique	None	❌ Will not implement
exists	None, use stat if necessary	❌ Will not implement

Action Plan, timeline, roadmap

This project will be achieved through the following steps:

1. Fully define and finalize the new API in WebIDL.
    a. ~~Once this API is approved by module owners, it will be merged into mozilla central.~~ Methods will be implemented one-by-one, along with introducing their IDL definitions.
    b. This will inform the incremental development which will occur through the summer.
2. Implement the new API in incremental patches.
    a. Submit a new patch per method or data structure, with tests
        - For methods of the new API which correspond directly to the OS.File API, ensure there are tests that assert compatibility between the two APIs. These tests will also assist in developing a migration guide.
    b. Do not introduce usages of the new API outside of tests until it is fully implemented.
3. Migrate the >1000 usages of OS.File in JavaScript to use the new IOUtils API described in this document.
    a. This will likely be done file-by-file in many patches.
4. Remove the implementation of OS.File entirely from Mozilla Central?

An approximate timeline is outlined below, with the understanding that this project is scoped to take my entire internship (mid May 2020, through to end of August 2020).

Work will be tracked on this Trello board: https://trello.com/b/hnIEePZu

Approx. Dates	Status	Work detail
May 14-22	Complete	Onboarding and in-depth review of existing OS.File javascript API, planning for the new API.
		OS.File API review
May 22-30	Complete	Draft IOUtils API specification in Web IDL
	Bug 1641699	Draft Web IDL specification with stubbed C++ implementation
June 1 - Aug 21	Complete	Implement API in C++
Aug 18 - Aug 28	In progress	Migrate OS.File API usages to use IOUtils
	Bug 986145	Stop using OS.File during startup
Later	Not started	Remove OS.File from mozilla-central

Extensions and future work

Consider a synchronous version of the API for Rust consumers.
https://bugzilla.mozilla.org/show_bug.cgi?id=1231711#c19

Consider adding streaming support.
This could use the W3C streams API.

Add an analogous interface for OS.File's directory iterator.
This will likely need async iterable support in webIDL.

A synchronous directory iterator would not be ideal, since it would probably need to hold an nsIFile in its instance.

An alternative approach is currently implemented by the method:
`Promise<sequence<DOMString>> IOUtils.getChildren(DOMString path)`

Questions

Open

None at the moment.

Answered

- We support iterable —Do we support WebIDL's `async iterable`? Would this be required to make an async directory iterator?

  A: IOUtils currently can't provide an async iterator like OS.File's DirectoryIterator object, since we don't support async iterables in WebIDL yet. However, we can achieve similar functionality with a call to `IOUtils.getChildren`. E.g.

  ```
  for (const dirEntry of await IOUtils.getChildren(path)) {
        /* do something */
  }
  ```

  Rather than:

  ```
  for await (const {path} of new DirectoryIterator(path)) {
        /* do something */
  };
  ```

- NSPR doesn't appear to have a method for creating a new hard or soft link. How do we extend NSPR to add support for this?
  - Is this even necessary, given `OS.File.unixSymlink` isn't used outside of tests?

  A: IOUtils will not provide an API for managing file links.

- How are platform differences handled in chrome code? Do we perform runtime checks and adjust behaviour accordingly (e.g. to use Windows Security Policies instead of Unix file mode/perms)?
  - If we implement on top of NSPR, do these runtime checks disappear?

  A: We'll perform platform checks sometimes, if it's deemed necessary for some performance enhancement, e.g. to make a call to `posix_fadvise` on Linux. We will have a single implementation though.

- NSPR/PRIO has *creationTime* (OS.File's corresponding file creationDate is buggy and deprecated, and most Linux file systems don't store creationTime in inodes).
    - Do we expose a creationDate on the new API?

  A: No, we will expose only what is available via the nsLocalFile C++ interface, since it is the only correct cross-platform way to represent/manage files. This interface only exposes modification times, and thus IOUtils will also only expose modification times.

- NSPR doesn't have a notion of "last access time"... Do we extend NSPR? Do we use something else to implement this?

  A: See the answer to the above question.