ARCHITECTURE DECISION BRIEF

Effect v4 in-memory FileSystem

Build a native, POSIX-rooted `MemoryFileSystem` layer behind the existing v4 contract, but establish conformance first and treat `memfs` as a prototype/reference—not code to import into Effect core.

22 JULY 2026 LOCAL EFFECT V4 BETA.100 · 6A0DCE96F7 DECISION SNAPSHOT

RECOMMENDATION
Portable native backend plus a first-class `Layer<FileSystem>`.

FIRST MOVE
Land a reusable conformance suite before the implementation.

NOT READY TO PORT
Neither earlier implementation satisfies the current 30-method contract.

Choose a native backend, reached through conformance

The long-term upstream shape should be a runtime-neutral module—most plausibly `effect/MemoryFileSystem`—whose `make` constructs isolated state and whose `layer` provides the existing `FileSystem.FileSystem` service. The backend should use explicit inodes, directory entries, descriptors, and symbolic links, while `FileSystem.make` continues to derive string, stream, sink, and existence helpers.

VERIFIED

The current interface exposes 30 operations, and `FileSystem.make` derives only five. Any usable replacement must truthfully implement the remaining 25 primitives; `layerNoop` is a stub, not an incremental base.

RECOMMENDATION

Do not begin by copying either prototype. Begin with observable scenarios shared by Node and memory implementations. This turns the current Node adapter into a bounded behavioral oracle and exposes where the public contract itself is ambiguous.

Inspect current FileSystem contract Inspect derived operations

The earlier efforts solve opposite halves

The closed Effect PR supplies a model without an adapter. `effect-memfs` supplies an adapter without a portable, current-v4 foundation. Their overlap is useful; neither is a merge candidate.

Inspected prior art and the current Effect repository

Source	What it proves	Material gaps	Reuse posture
effect-smol PR #456	Pure inode and descriptor model; 37 focused tests; common POSIX error cases.	No service or layer; partial operations; broken closed-handle, relative-symlink, unlink lifetime, permission, and time semantics.	Reuse concepts and adversarial cases. Its MIT-hosted provenance is clear.
effect-memfs	A Node-style filesystem can be adapted into Effect with scoped handles, seeded volumes, temp resources, copy, links, and watch.	Effect v3; no `glob`; Node builtins and `Buffer`; weak cancellation/watch tests; no snapshot control surface.	Use as architecture reference or a fast external prototype. Do not copy source without licensing clarification.
Current v4 repository	Stable service boundary, derived helpers, normalized `PlatformError`, Node behavior, and open-handle expectations.	Only nine direct Node tests; no reusable conformance suite; several semantics are undocumented.	Make this the source of truth.

Inspect PR #456 Inspect its only implementation commit Inspect effect-memfs

▶ **Specific defects found in PR #456**

A separate memfs adapter is the fast fallback, not the upstream target

Three options are viable, but they optimize for different horizons. The native option best matches the stated goal of an Effect v4 implementation usable in tests, CLIs, and eventually browsers.

Options assessed against portability, delivery, conformance, and maintenance

Option	Delivery	Portability	Semantic ownership	Decision
Native `effect/MemoryFileSystem`	Slowest initial slice	Node, Bun, browser, workers	Effect owns and tests the contract	**Recommend** for upstream
New `@effect/platform-memory` **backed by memfs**	Fast	Potentially browser-capable after removing Node coupling	Delegated to memfs plus adapter quirks	Good fallback if core size is rejected
External v4 effect-memfs port	Fastest for stack-effect	Node/Bun first	Outside Effect's conformance guarantees	Useful dogfood prototype

LICENSE BOUNDARY

`nounder/effect-memfs` has no declared source or package license. Its implementation should not be copied into Effect without explicit permission or a license. The underlying `memfs` dependency is Apache-2.0.

Extract models, adapter lessons, and tests—not files

From PR #456

The inode/directory-entry split; separate per-open descriptors; link counts; bounded symbolic-link resolution; error-focused test cases.

From effect-memfs

The `make` / `layer` / `layerWith` ergonomics; scoped descriptor close; independent append/read cursor behavior; seeded volumes; watcher path normalization.

From the Node adapter

The current v4 method signatures, normalized errors, open flags, cursor truncation behavior, temp-resource finalizers, and `FileSystem.make` boundary.

Do not extract yet

A snapshot/diff/apply API. That belongs to a later memory-specific control service or to stack-effect's Plan-to-Apply boundary, not the platform-neutral `FileSystem` contract.

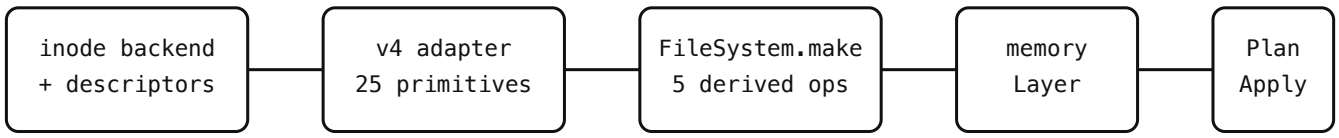
Keep the public layer small and the storage model honest

The public module needs very little surface. Complexity belongs behind the adapter, not in a new virtual-tree abstraction.

```
import { Effect, FileSystem, Layer } from "effect"
import * as MemoryFileSystem from "effect/MemoryFileSystem"

const program = Effect.gen(function*() {
  const fs = yield* FileSystem.FileSystem
  yield* fs.makeDirectory("/app/src", { recursive: true })
  yield* fs.writeFileString("/app/src/index.ts", "export {}\n")
})

Effect.provide(program, MemoryFileSystem.layer)
```



The in-memory layer stops at filesystem behavior. Conflict resolution and durable application remain explicit downstream stages.

The internal model should provide explicit directory entries, shared inode data for hard links, raw-target symbolic links, per-layer file-descriptor counters, defensive byte copies, virtual POSIX root `/`, per-handle cursors and access modes, and deterministic time through Effect's `Clock`.

Test the v4 contract before claiming POSIX compatibility

The first suite should target behavior needed by scaffolding and the existing API. “POSIX-compliant” is too broad for a first PR and conflicts with existing Effect behavior in places.

Prioritized observable behavior

Priority	Area	Required proof
P0	Isolation and bytes	Layer builds do not share state; one build does; write/read values cannot alias stored bytes.
P0	Scaffold operations	Recursive mkdir, read/write, stat, readDirectory, rename, copy, truncate, recursive remove.
P0	Flags and handles	All ten flags; scoped closure; independent cursors; append; EOF; sparse writes; rename/unlink with open handles.
P0	Errors	Exact outer `PlatformError` plus reason tag, method, and original path/descriptor.
P0	Derived operations	Streams with offsets/limits/chunks, sink truncation, string encoding, and finalization.
P1	Links and metadata	Relative/absolute symlinks, loops, hard links, link counts, `realPath`, `TestClock` time transitions.
P1	Temp, glob, copy	Prefixes/suffixes, scoped cleanup, overwrite, preserved time, roots and excludes.
P2	Watch and properties	Deterministic registration, event ordering, subscriber cleanup, random operation invariants.

Land the work as four reviewable slices

1. **Contract suite.** Extract stable scenarios and run them against Node. Resolve the seek/cwd/watch-path questions exposed by tests.
2. **Scaffold-capable layer.** Add isolated state, paths, files/directories, exact errors, all open flags, handles, streams, sinks, and temp resources.
3. **Deep filesystem semantics.** Add links, metadata/time, glob, copying fidelity, and deterministic concurrency checks.
4. **Watching and control.** Add watch only with a registration handshake; separately evaluate seed/snapshot export for stack-effect.

SMALLEST USEFUL MILESTONE

Slice two is enough to swap the layer under stack-effect's scaffold generation. Disk reconciliation can remain outside the layer while the filesystem conformance surface matures.

Resolve five contract questions during the first slice

OPEN

> **Module placement:** `effect/MemoryFileSystem` is the recommendation; a separate `@effect/platform-memory` is the fallback if dependency or core-size concerns dominate.

- Should `File.seek` return the new cursor? Its type says `void`, while Node returns and tests assert the cursor.
- Should relative paths always resolve from virtual `/`, or should `make` accept a virtual cwd?
- Are watch event paths relative to the watched path or canonical absolute virtual paths?
- Should `access` only prove existence until a virtual identity model exists?
- Should memory handle truncation match Effect's current Node cursor-clamping behavior rather than POSIX `ftruncate`?

Evidence boundary

This report inspected the linked PR, its commit/specifications/tests, the current effect-memfs source/tests/history/package metadata, stack-effect issue #108, and the local v4 FileSystem, Node adapter, error model, and tests. No implementation was added and no Effect tests were run in this exploration. Repository and package status are a 22 July 2026 snapshot.

`Inspect stack-effect issue #108` `Inspect published effect-memfs package`

Effect v4 in-memory FileSystem · architecture decision brief · 22 July 2026