Zaxon Working Group
Request for Discussion and Implementation Record

ZDS 0006
Format Contract

Windows Durability and the Supported Platform Floor

STATE
published
CREATED
2026-07-25
LAST UPDATED
2026-07-25

INTENDED STATUS
Normative
AUTHORS
paxos-zig project

DISCUSSION

How a pathname transition is made durable where there is no directory fsync

LABELS
architecture, zaxonlite, durability, platforms

Status of This Memo

This document is an internal Zaxon discussion record authored in Typst and tracked in git. It intentionally follows RFC-style structure so the design scope, rationale, trade-offs, and operational constraints remain explicit. Documents that still use the placeholder number XXXXX are provisional drafts. Numbered ZDS documents are part of the project record.

Contents

1. Abstract

Zaxonlite's storage contract requires that an authoritative file survive a crash together with its directory entry. On POSIX this is one call: sync the parent directory after the rename. Windows offers no directory sync, which is why the platform was excluded from the first release.

It needs none. NTFS records namespace changes in a volume-wide write-ahead log, and flushing a file commits that log. The transition is therefore made durable by flushing the file it produced, which means the barrier follows the rename where POSIX puts it before. This record states that model, the floor it requires, how the floor is enforced, and what remains unproven.

2. Terminology and Scope

A *pathname transition* is an authoritative create, link, rename, or removal: journal creation, payload installation, snapshot publication, the `CURRENT` pointer, identity and enrollment files, backups. A *barrier* is a sync that must complete before anything is acknowledged, voted, or sent.

In scope: how a pathname transition is persisted on Windows, and the minimum platform that supports it. Out of scope: unix socket listeners, the Windows test harness, and the payload-write optimization noted at the end.

3. Problem Statement

ZDS 0004 froze the supported matrix as POSIX systems only, and the durability layer returned an error on Windows rather than weakening the guarantee. That was the correct default, but it rested on an incomplete reading. Two claims deserve separating:

1. Windows cannot flush a directory handle the way POSIX can. This is true and is not going to change.
2. Windows therefore cannot persist a directory entry. This does not follow.

4. The Model

NTFS is write-ahead logged. `$LogFile` is a single sequential metadata journal per volume, so flushing it through a given record persists every record before it. Flushing a file forces the log through that file's last update. Any file flush therefore persists every earlier namespace transition on the same volume.

Microsoft documents the file, rather than the directory, as the durable unit. From the `CreateFile` caching-behavior reference: a write-through request "also causes NTFS to flush any metadata changes, such as a time stamp update or a rename operation, that result from processing the request"; and, directly on point for journal creation, "the file metadata may still be cached (for example, when creating an empty file). To ensure that the metadata is flushed to disk, use the FlushFileBuffers function."

Three consequences follow, and all three are load-bearing.

1. **The barrier moves after the transition.** POSIX syncs the file, renames, and then syncs the parent. Windows syncs the file, renames, and then syncs the file again. A directory sync that simply became a no-op, with the call sites left alone, would leave nothing flushed after the rename. That change compiles, passes the crash matrix – process death loses nothing under any sync mode – and loses acknowledged writes on power loss. It is precisely the failure this record exists to prevent.
2. **A no-op directory sync is correct only where a later barrier is named.** This is the existing `#raw("...BeforeBarrier")` contract, which already delegates persistence to the

caller's next barrier. Both snapshot renames qualify: a snapshot becomes authoritative only when the stop sign naming it is journaled and synced, and an installed snapshot is followed immediately by the `CURRENT` pointer write. Transitions with no such successor – enrollment tokens, published identities, backups, journal creation – take their barrier at once through `syncPathnameTransition`.

3. **The sync policy is unchanged.** A normal Windows flush already synchronizes the underlying storage cache, so it is `F_FULLFSYNC`-strength and the `os` and `full` modes are equivalent there, as they already are on Linux.

5. The Platform Floor

Windows 10 release 1809 or Windows Server 2019 and newer, on NTFS.

The version is set by rename semantics rather than by durability. Zig's Windows rename issues `FILE_RENAME_INFORMATION_EX` with POSIX semantics and replace-if-exists, which is what allows an open file to be replaced, and it sets the ignore-read-only-attribute flag alongside. That flag requires 1809; an unsupported flag is refused as a unit, so releases from 1607 to 1803 fall back silently to the legacy rename and lose the semantics the storage layer assumes. Windows 10 reached end of life in October 2025, so nothing is lost by requiring the higher floor.

The filesystem requirement is not tidiness. FAT has no metadata log at all, so the argument above does not weaken there, it disappears. ReFS is excluded as untested: it is logged, but its allocate-on-write metadata model differs and the ordering property has not been established for it.

6. Enforcement

The floor is enforced by behavior, not by a version check. Two facts force this. The standard library exposes no version API, and the rename fallback is silent – the call reports success either way – so a version check would test a proxy rather than the property.

A node therefore probes at startup, under the data-directory lock and before any storage exists: create a file, hold it open, create a second, rename the second over the first. Replacing an open file is exactly what the legacy path cannot do, so one operation rejects old Windows, FAT volumes, and network filesystems that degrade quietly, without naming any of them. POSIX platforms pass it, where it also catches an exotic mount.

7. Amendments to ZDS 0004

The supported production matrix in ZDS 0004 is amended: Windows is supported at the floor above, under the model in this record. Formats, wire protocol, and every other clause of that record are unaffected – nothing here changes a durable or on-the-wire byte.

Unix socket listeners remain POSIX-only. Windows has AF_UNIX but attaches no file mode to it; access is governed by a DACL that the existing permission narrowing cannot express, so binding one would produce a socket reachable by every local account. The listener is rejected during configuration.

8. What Remains Unproven

Two limits belong in the record rather than in a footnote.

The log-ordering property – that flushing one file persists earlier namespace transitions on the volume – is standard write-ahead-log behavior and is consistent with the documentation quoted above, but it is an inference from NTFS's design rather than a documented contract. It should be treated as the weakest link in this record.

No test suite runs on Windows. Cross-compilation is gated in CI, which proves the port builds and links, not that it works. A Windows runner is the next step, and this record stays at `published` rather than `committed` until one exists. Power-loss durability is untested on every platform, Windows included; that limit is not new here.

9. Future Work

Payload writes currently move bytes with a non-barrier flush and rely on the journal barrier that follows. Windows has the same split available through `FLUSH_FLAGS_NO_SYNC`, which writes cached data and metadata without synchronizing the storage cache. It is deliberately not used yet: the first implementation stays on the portable interface, and the optimization should arrive with a benchmark that justifies it, as the macOS full-flush policy did.