ProjFS Managed API

From C++/CLI to Pure C#

Drop the C++ toolchain. Keep the API.
Enable NativeAOT for all ProjFS consumers.

Why Change?

The current `ProjectedFSLib.Managed.dll` is a **C++/CLI mixed-mode assembly**.

Problem	Impact
Requires VS C++ workload + C++/CLI support	Build infrastructure complexity
Requires Visual C++ redistributable	Deployment/installer overhead
Incompatible with NativeAOT	Blocks modern .NET optimization
Incompatible with trimming	Larger deployment sizes
Targets netcoreapp3.1 / net48	Stuck on legacy TFMs
Two languages in one project	Higher maintenance bar

What We Did

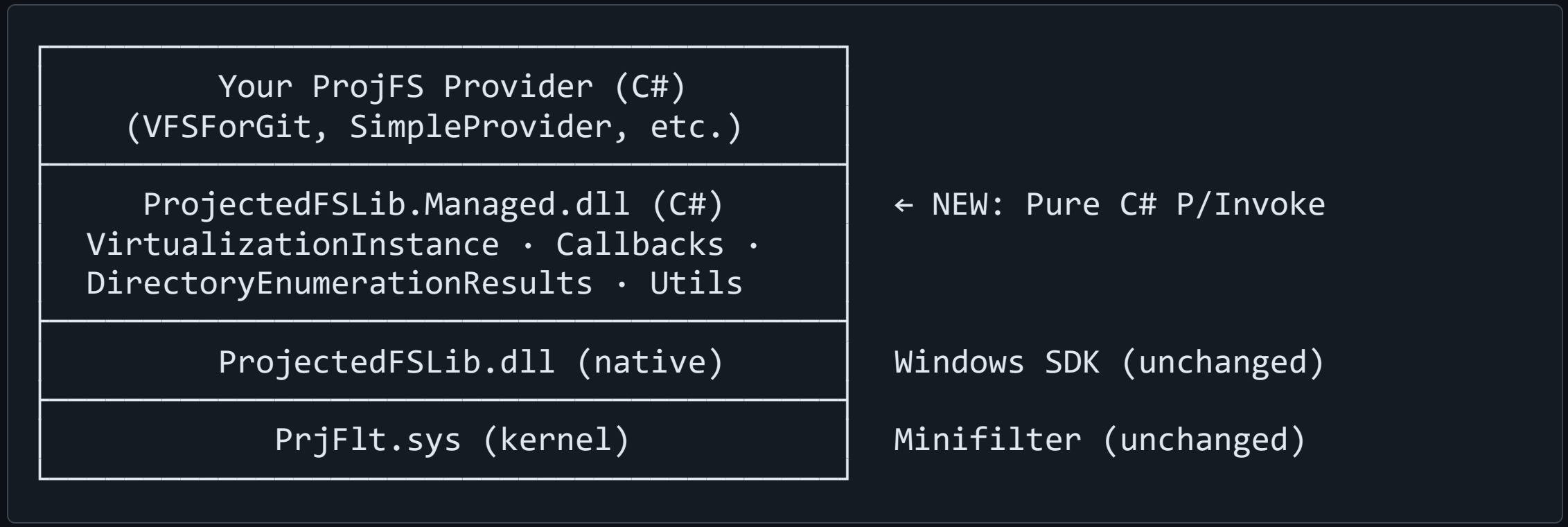
Pure C# P/Invoke replacement — same namespace, same API, zero C++.

```
<!-- Before -->
<PackageReference Include="Microsoft.Windows.ProjFS" Version="1.2.19351.1" />

<!-- After -->
<ProjectReference Include="ProjectedFSLib.Managed.CSharp.csproj" />
<!-- No other code changes needed -->
```

- `Microsoft.Windows.ProjFS` namespace preserved
- All types, interfaces, delegates, enums — identical signatures
- 16/16 existing tests pass (including symlink tests)
- **netstandard2.0** target for .NET Framework 4.8 / .NET Core 3.1 compatibility

Architecture



We replaced **one layer** — the managed wrapper. Everything above and below is unchanged.

Key Technical Decisions

Struct Layout Verification

Every P/Invoke struct verified against Windows SDK with native `sizeof()` / `offsetof()`:

- `PRJ_PLACEHOLDER_INFO` = **344 bytes** (includes `VariableData[1]` flexible array)
- Missing 8 bytes caused `ERROR_INSUFFICIENT_BUFFER` for all placeholder writes

String Marshaling

`PCWSTR` fields use `Marshal.StringToHGlobalUni()`, **not** `GCHandle.Alloc + AddrOfPinnedObject`.

Pinning a .NET string gives the object header address, not the character data.

Notification Mapping Lifetime

ProjFS caches notification mapping pointers — must keep alive for instance lifetime.

What We Learned

ReFS Does Not Support ProjFS Symlinks

`WritePlaceholderInfo2` (symlink placeholders) fails with `ERROR_NOT_SUPPORTED` on ReFS volumes.

Root cause (via WinDbg + kernel source):

- `PrjFlt.sys` → `PrjfCreateSymbolicLink` → `FltCreateFileEx2` with atomic create ECP
- NTFS: ✅ Supports `GUID_ECP_ATOMIC_CREATE` with `IO_REPARSE_TAG_SYMLINK`
- ReFS: ❌ Returns `STATUS_NOT_SUPPORTED` (`0xC00000BB`)

Non-symlink operations work on both NTFS and ReFS.

Results

Metric	C++/CLI	Pure C#
Build toolchain	VS 2022 + C++ + C++/CLI	`dotnet build`
Runtime deps	VC++ Redist + ljwhost.dll	None
NativeAOT	❌	✅
Trimming	❌	✅ (`IsAotCompatible=true`)
TFMs	net48, netcoreapp3.1	netstandard2.0, net8.0, net9.0, net10.0
Tests passing	16/16	16/16
Lines of code	~4,000 (C++/CLI)	~1,700 (C#)
Source files	30+ (.h, .cpp, .vcxproj)	5 (.cs, .csproj)

Migration Path for Consumers

1. Replace NuGet package reference with project reference
2. Remove `<UseIJWHost>True</UseIJWHost>`
3. Remove VC++ redistributable from installer
4. **No code changes** — same namespace, same API

For VFSForGit specifically:

This unblocks the complete .NET 10 + NativeAOT migration:

- 3.5x faster startup (53ms → 15ms)
- 2x faster pipe roundtrips (341ms → 179ms)
- 78% faster file reads
- Zero regressions across all benchmarks

Ask

1. **Accept the PR** to replace C++/CLI with pure C# in ProjFS-Managed-API
2. **Publish updated NuGet** targeting modern .NET TFMs
3. **VFSForGit** can then depend on the upstream package instead of vendoring

This is **incremental** — the old C++/CLI package continues to work for existing consumers.
New consumers get NativeAOT support, simpler builds, and fewer dependencies.