

ATHEM ENGINE

Architecture & Implementation Guide

Project Structure, Design Patterns, and Technical Reasoning

Version 1.0 | December 2025

1. Architectural Philosophy

1.1 Core Principles

Principle 1: USD as the Single Source of Truth

Every piece of world state—transforms, properties, relationships, animations—lives in the OpenUSD Stage. No game state exists outside USD. This ensures serialization, undo/redo, and tool interoperability are automatic.

Rationale: Game engines traditionally scatter state across multiple systems (transform cache, physics world, render scene). This creates synchronization bugs and makes serialization complex. USD's composition engine already solves these problems.

Principle 2: Computation Flows Through OpenExec

All derived values (world matrices, LOD levels, visibility flags, animation evaluations) are computed via OpenExec's dependency graph. No manual dirty-flagging or cache invalidation.

Rationale: Manual cache invalidation is the source of countless bugs in game engines. OpenExec's DAG-based approach guarantees correctness while enabling parallel evaluation.

Principle 3: Layer-Based with Feature Cohesion

Top-level organization follows infrastructure layers (Core, Exec, Runtime, Render, Physics). Within each layer, code is organized by feature.

Rationale: Pure layer-based scatters related code. Pure feature-based creates circular dependencies. The hybrid matches how Pixar structures USD/Hydra while keeping features discoverable.

Principle 4: MVVM for Editor, Data Pipeline for Runtime

The editor uses MVVM with USD as the backing store. The runtime uses a direct data pipeline (USD → OpenExec → Hydra). Different problems, different patterns.

Principle 5: Platform Abstraction at Leaf Nodes

Platform-specific code (Metal vs Vulkan, dylib vs Wasm) lives in leaf modules behind protocols. Core systems never import platform headers directly.

1.2 SwiftUSD Integration Considerations

Athem uses AthemIO/SwiftUSD as its USD foundation. This library provides native Swift bindings to OpenUSD 25.11 via C++ interoperability. The following considerations apply:

C++ Interop Mode Required

All targets importing PixarUSD must enable Swift/C++ interoperability. This is configured in Package.swift with .interoperabilityMode(.Cxx) and cxxLanguageStandard: .cxx17.

Pixar.Bundler Initialization

USD resources must be initialized before any USD operations. Call Pixar.Bundler.shared.setup(.resources) at application startup in the Athem executable, before initializing any engine subsystems.

Plugin Registration

Custom USD plugins (Schema, ExecPlugins) must be registered after Bundler setup using Pixar.PlugRegistry. The PluginLoader in Sources/Athem/Configuration/ handles discovery and registration of engine plugins.

Namespace Conventions

SwiftUSD exposes USD types under the Pixar namespace (e.g., Pixar.Usd.Stage, Pixar.UsdGeom.Xform). Engine code should use these directly rather than creating wrapper types unless abstraction is necessary.

Platform Support

SwiftUSD supports macOS (.v14+), iOS (.v17+), visionOS (.v1+), Linux, and Windows. Platform-specific rendering backends (Metal, Vulkan) are handled in the Render module.

swift-bundler for Builds

SwiftUSD recommends swift-bundler for building applications. Athem uses Bundler.toml for build configuration, which aligns with SwiftUSD's tooling. Use 'swift bundler run' for development and 'swift bundler bundle' for distribution.

2. High-Level Project Structure

```
Athem/
├── Package.swift          # Swift Package Manager manifest
├── Bundler.toml           # Swift Bundler configuration

└── Sources/                # Swift source code
    ├── Athem/               # Main executable
    ├── Core/                # Engine foundation
    ├── Exec/                # OpenExec integration
    ├── Runtime/              # Script runtime (Native/Wasm)
    ├── Render/               # Hydra rendering
    ├── Physics/              # Jolt physics
    ├── Audio/                # Audio system
    ├── AI/                  # LLM integration
    ├── Editor/               # Editor UI (MVVM)
    └── SDK/                 # Public scripting API

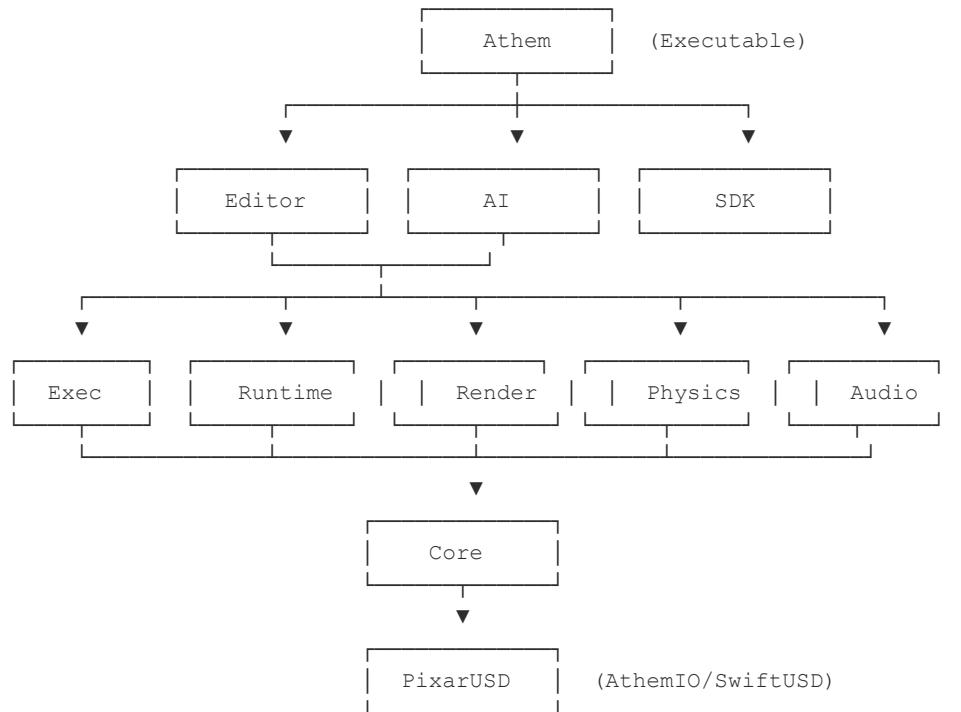
└── Plugins/                # C++ USD/OpenExec plugins
    ├── Schema/              # Custom USD schemas
    └── ExecPlugins/         # OpenExec computations

└── Tests/                  # Unit and integration tests
└── Resources/              # Shaders, default assets
└── Templates/              # Project templates
└── Documentation/          # Guides and API docs
└── Tools/                  # Build scripts
```

└ Examples/ # Sample projects

3. Module Dependency Graph

Dependencies flow downward. No cycles. Each module only imports what it needs.



3.1 Dependency Rules

1. **Core** depends only on **PixarUSD** (AthemIO/SwiftUSD). Provides engine fundamentals.
2. **Exec** depends on **Core**. Wraps OpenExec computation APIs.
3. **Runtime** depends on **Core**. Script runtime abstraction (Native/Wasm).
4. **Render** depends on **Core** and **Exec**. Hydra/Storm bridge.
5. **Editor** depends on all subsystems. MVVM UI layer.
6. **SDK** depends on **Core** only. Clean public API for users.

4. Complete Folder Structure

4.1 Athem (Executable)

```
Sources/Athem/
└── main.swift                      # Application entry point
    └── AppDelegate.swift           # Must call Pixar.Bundler.shared.setup(.resources)
        └── Configuration/
            ├── LaunchConfiguration.swift # Platform app lifecycle
            └── PluginLoader.swift      # Registers USD plugins via PlugRegistry
```

4.2 Core

```
Sources/Core/
└── Engine/
    ├── Engine.swift                # Main engine, owns subsystems
    ├── GameLoop.swift              # Fixed timestep game loop
    └── TimeManager.swift           # Delta time, game time
└── Stage/
    ├── StageManager.swift          # USD Stage lifecycle
    ├── PrimRegistry.swift          # Fast prim lookup
    └── StageNotifier.swift         # Change notification
└── Input/
    ├── InputManager.swift          # Input manager
    └── Devices/
        ├── KeyboardDevice.swift     # Keyboard device
        ├── MouseDevice.swift        # Mouse device
        └── GamepadDevice.swift       # Gamepad device
└── Services/
    ├── ServiceLocator.swift        # Service locator
    └── LogService.swift            # Log service
```

4.3 Exec

```
Sources/Exec/
└── Core/
    ├── ExecSystem.swift           # ExecUsdSystem wrapper
    ├── ComputeRequest.swift       # Batched requests
    ├── ValueKey.swift             # Computation identifier
    └── CacheView.swift            # Results access
└── Features/
    ├── Transform/
        └── TransformComputation.swift
    ├── LOD/
        ├── LODComputation.swift
        └── LODConfig.swift
    ├── Culling/
        ├── BoundsComputation.swift
        └── VisibilityComputation.swift
    └── Animation/
        └── SplineComputation.swift
```

4.4 Runtime

```
Sources/Runtime/
└── Core/
    ├── GameScriptRuntime.swift     # Protocol
    └── HostInterface.swift        # Script ↔ USD bridge
```

```
|   └── RuntimeManager.swift
| └── Native/                      # Desktop runtime
|   ├── NativeRuntime.swift
|   ├── DylibLoader.swift
|   └── HotReloader.swift
└── Wasm/                         # iOS/sandboxed runtime
   ├── WasmRuntime.swift
   ├── WasmHostFunctions.swift
   └── WasmBridge.swift
```

4.5 Render

```
Sources/Render/
├── Core/
│   ├── HydraEngine.swift
│   ├── SceneIndexBridge.swift
│   └── RenderSettings.swift
├── Viewport/
│   ├── ViewportProtocol.swift
│   ├── Metal/
│   │   └── MetalViewport.swift
│   ├── Vulkan/
│   │   └── VulkanViewport.swift
│   └── OpenGL/
│       └── OpenGLViewport.swift
└── Camera/
    ├── CameraController.swift
    └── Frustum.swift
```

4.6 Editor (MVVM Architecture)

```
Sources/Editor/
├── App/
│   ├── EditorApp.swift
│   ├── EditorCoordinator.swift
│   └── EditorState.swift
├── Core/
│   ├── Bindings/
│   │   ├── USDBinding.swift
│   │   ├── StageObserver.swift
│   │   └── SelectionBinding.swift
│   ├── Commands/
│   │   ├── Command.swift
│   │   ├── CommandStack.swift
│   │   └── Commands/
│   │       ├── CreatePrimCommand.swift
│   │       ├── TransformCommand.swift
│   │       └── PropertyCommand.swift
│   └── Services/
│       ├── SelectionService.swift
│       └── ClipboardService.swift
└── Features/
    ├── Hierarchy/
    │   ├── Models/
    │   │   └── HierarchyItem.swift
    │   ├── ViewModels/
    │   │   └── HierarchyViewModel.swift
    │   └── Views/
    │       └── HierarchyPanel.swift
    ├── Inspector/
    │   ├── Models/
    │   │   └── PropertyDescriptor.swift
    │   ├── ViewModels/
    │   │   ├── InspectorViewModel.swift
    │   │   ├── TransformVM.swift
    │   │   └── GameAPIVM.swift
    │   └── Views/
    │       ├── InspectorPanel.swift
    │       └── TransformSection.swift
    ├── Viewport/
    │   ├── Models/
    │   ├── ViewModels/
    │   │   ├── ViewportViewModel.swift
    │   │   └── GizmoVM.swift
    │   └── Views/
    │       ├── ViewportPanel.swift
    │       └── Overlays/
    │           └── GizmoOverlay.swift
    ├── Assets/
    ├── Timeline/
    ├── Console/
    └── AIAssistant/
Shared/
└── Components/
    ├── PropertyFields/
    │   ├── FloatField.swift
    │   ├── Vector3Field.swift
    │   └── ColorField.swift
```

```
|   └── Common/
|       └── CollapsibleSection.swift
└── Extensions/
```

4.7 SDK

```
Sources/SDK/
├── Core/
│   ├── GameScript.swift          # Base class for user scripts
│   ├── Entity.swift
│   └── World.swift
├── Components/
│   ├── Transform.swift
│   ├── MeshRenderer.swift
│   ├── Rigidbody.swift
│   └── AudioSource.swift
├── Math/
│   ├── Vector3.swift
│   ├── Quaternion.swift
│   └── Matrix4.swift
├── Input/
│   └── Input.swift
└── Physics/
    └── Physics.swift
```

4.8 C++ Plugins

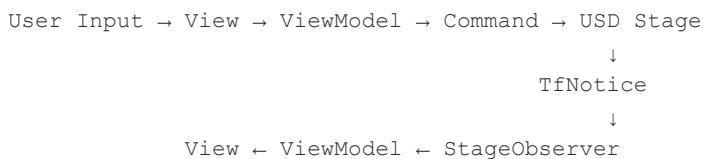
```
Plugins/
├── Schema/                      # Custom USD schema
│   ├── CMakeLists.txt
│   ├── schema.usda
│   └── plugInfo.json
└── Sources/
    ├── gameAPI.h
    ├── gameAPI.cpp
    └── tokens.cpp
└── ExecPlugins/                 # OpenExec computations
    ├── CMakeLists.txt
    ├── plugInfo.json
    └── Sources/
        ├── execRegistration.cpp
        ├── lodComputation.cpp
        └── boundsComputation.cpp
```

5. Data Flow Analysis

5.1 Runtime Frame Flow

1. Poll Input → InputManager
2. Run Scripts → RuntimeManager → writes to USD
3. Step Physics → PhysicsWorld → writes transforms to USD
4. OpenExec → Automatic invalidation + recompute
Compute (World transforms, LOD, bounds)
5. Hydra Sync → Scene index update
6. Storm Render → Metal/Vulkan/OpenGL

5.2 Editor MVVM Flow



6. Design Pattern Summary

Pattern	Where Used	Why
MVVM	Editor	Reactive UI. ViewModels are testable. USD serves as Model.
Command	Editor Commands	Encapsulates mutations. Enables undo/redo stack.
Strategy	Runtime	GameScriptRuntime protocol. Native/Wasm swappable.
Observer	StageObserver	USD TfNotice broadcasts changes to ViewModels.
Bridge	PhysicsBridge	Connects USD and Jolt without coupling.
Protocol	ViewportProtocol	Platform abstraction. Metal/Vulkan/OpenGL implement same interface.
DAG	OpenExec	Computations form dependency graph. Auto invalidation.

7. Scaling Guidelines

7.1 Adding a New Computation

1. Define callback in Plugins/ExecPlugins/Sources/
2. Register with EXEC_REGISTER_COMPUTATIONS_FOR_SCHEMA
3. Rebuild C++ plugin
4. Create Swift wrapper in Exec/Features/<Name>/

7.2 Adding a New Editor Panel

1. Create folder: Editor/Features/<PanelName>/
2. Add Models/, ViewModels/, Views/ subfolders
3. Create @Observable ViewModel observing StageObserver

4. Build View using Shared/Components/
5. Register in EditorCoordinator

7.3 Adding a New Platform

1. Implement ViewportProtocol in Render/Viewport/<Platform>/
2. Add platform input device in Core/Input/Devices/
3. Update Package.swift with platform conditionals
4. Add shaders in Resources/Shaders/<Platform>/

7.4 Adding a New SDK Component

1. Create component in SDK/Components/
2. Map to USD schema attributes
3. Ensure Native + Wasm compilation works
4. Add inspector section in Editor

8. Package.swift Configuration

Reference configuration for integrating AthemIO/SwiftUSD:

```
// swift-tools-version: 6.0
import PackageDescription

let package = Package(
    name: "Athem",
    platforms: [
        .macOS(.v14),
        .iOS(.v17),
        .visionOS(.v1)
    ],
    dependencies: [
        .package(url: "https://github.com/AthemIO/SwiftUSD.git", from: "25.11.0"),
    ],
    targets: [
        .executableTarget(
            name: "Athem",
            dependencies: ["Core", "Editor", "Runtime", "Render"],
            swiftSettings: [.interoperabilityMode(.Cxx)]
        ),
        .target(
            name: "Core",
            dependencies: [.product(name: "PixelUSD", package: "SwiftUSD")],
            swiftSettings: [.interoperabilityMode(.Cxx)]
        ),
        // ... other targets follow same pattern
    ],
    cxxLanguageStandard: .cxx17
)
```

8.1 Critical Swift Settings

- **.interoperabilityMode(.Cxx):** Required for all targets that import PixarUSD or depend on targets that do.
- **cxxLanguageStandard: .cxx17:** USD requires C++17. Must be set at package level.
- **Windows-specific defines:** Add _ALLOW_COMPILER_AND_STL_VERSION_MISMATCH and _ALLOW_KEYWORD_MACROS for Windows builds.

8.2 Initialization Sequence

```
// main.swift - Application entry point
import PixarUSD
import Core

@main
enum Athem {
    static func main() {
        // 1. Initialize USD resources (MUST be first)
        Pixar.Bundler.shared.setup(.resources)

        // 2. Register custom plugins
        PluginLoader.registerEnginePlugins()

        // 3. Initialize engine
        let engine = AthemEngine()
        engine.run()
    }
}
```

```
 }  
}  
— End of Document —
```