Bedrock Architecture Specification v1.0

Version: 1.0 (Genesis)

Date: December 30, 2025

Status: CONSTITUTIONAL DOCUMENT

Branch: `bedrock`

Relationship: Strangler fig replacement for `/foundation` (legacy)

Preamble: The Strangler Fig Pattern

This document establishes the architecture for **Bedrock**—a clean implementation of Grove's administrative and exploration surfaces built from first principles.

What we're NOT doing: Refactoring existing `/foundation` code.

What we ARE doing: Building a new system alongside the old, proving it works, then deprecating legacy.

The Bifurcation

Aspect	`/foundation` (Legacy)	`/bedrock` (New)
Status	FROZEN (bugs only)	Active development
Purpose	Powers Genesis marketing	Reference implementation
Patterns	Mixed, evolved organically	Consistent, DEX-native
Copilot	Partial, inconsistent	Universal, context-aware
Timeline	Deprecated when Bedrock feature-matches	Ships when ready

Part 1: Constitutional Foundation

Bedrock implements the principles established in Grove's founding documents. These are **not suggestions**—they are constraints that gate all architectural decisions.

The Trellis Architecture: First Order Directives

> *"Models are seeds. Exploration architecture is soil."*

Every Bedrock component must satisfy the **DEX Stack Standards**:

I. Declarative Sovereignty

Can a non-technical domain expert alter behavior by editing a config file?

- If NO → feature is incomplete
- All console behaviors configurable via JSON/YAML
- No exploration paths hardcoded in TypeScript

II. Capability Agnosticism

Does the system break if the model hallucinates?

- If YES → architecture is incomplete
- Trellis catches errors; models provide intelligence
- Schema validation rejects invalid patches

III. Provenance as Infrastructure

Can every fact trace back to its origin?

- If NO → it's a bug
- All Sprouts maintain attribution chains
- Version history tracks who/what modified objects

IV. Organic Scalability (The Trellis Principle)

Does structure support growth without inhibiting it?

- Guided wandering, not rigid tunnels
- New object types inherit all infrastructure automatically
- Extension through configuration, not code changes

Cross-Reference: Founding Documents

Document	What It Governs	Bedrock Must...
The_Trellis_Architecture__First_Order_ Directives.md	Philosophy, DEX principles	Pass all four tests
Kinetic_Framework_Strategic_ Vision.md	Bigger picture, MCP/Proto-Skills	Enable framework extraction
copilot-configurator-vision.md	AI assistant pattern	Implement universal Copilot
grove-object.ts	Data model	Use GroveObject for all entities
GROVE_FOUNDATION_REFACTOR_ SPEC.md	Feature inventory	Implement all proven features

Part 2: The Object Model Boundary

The most critical architectural insight from our audit:

Sprouts = Knowledge Lifecycle (Write Path)

Every content change flows through the Sprout process:

```
Document upload → Mason processes → Sprout (tender)
Claim extraction → Sprout claims (rooting)
Contradiction detected → Sprout flag for resolution
Research suggestion → Sprout proposal
Human validation → Sprout promotion
Outdated content → Sprout deprecation
```

Sprout is the atomic unit of knowledge change.

DEX Objects = Processing Infrastructure (Read Path)

Organize how validated Sprouts are navigated and surfaced:

Object	Purpose	Relationship to Sprouts
Hub	Topological container	Where validated Sprouts live
Journey	Navigation path	How users traverse Sprout clusters

Object	Purpose	Relationship to Sprouts
Node	Journey waypoint	Surfaces specific Sprouts
Lens	Personalization filter	Which Sprouts to emphasize
Moment	Engagement trigger	When to surface Sprouts
Persona	User archetype	Sprout presentation style

System Objects = Global Configuration

- Feature Flags: Runtime toggles
- System Voice: Base personality configuration
- Health Checks: Operational status
- Audio Tracks: Media assets

This boundary is inviolable. Consoles are organized around these categories, not arbitrary feature groupings.

Part 3: The Canonical Console Pattern

Every Bedrock console follows this exact structure:

```
┌──────────────────────────────────────────────────────────
─────────────┐
│ Console Header (title, description, actions)          │
├──────────────────────────────────────────────────────────
─────────────┤
│ Metrics Row (4-6 key stats, always visible)           │
├──────────────┬──────────────────────────────┬──────────────
────────────┤
│ Navigation │ Content Area        │ Inspector + Copilot  │
│ (240px)    │ (flex)              │ (360px)              │
│            │                     │                      │
│ Object     │ Grid/List/Editor    │ Detail View          │
│ Collection │ (context-dependent) │ + AI Suggestions     │
│            │                     │ + Quick Actions      │
│            │                     │ + Version History    │
```

```
└────────────────────┴────────────────────────────────────────┴───────────────────
────────────────┘
```

BedrockLayout (The Primitive)

```
// src/bedrock/primitives/BedrockLayout.tsx

interface BedrockLayoutProps {
  // Identity
  consoleId: string;
  title: string;
  description: string;
  icon: string;

  // Metrics (always visible)
  metrics: MetricCardProps[];

  // Three columns
  navigation: ReactNode;
  content: ReactNode;

  // Inspector integration
  inspectorConfig: InspectorConfig;

  // Header actions
  actions?: ConsoleAction[];

  // Copilot configuration
  copilot: CopilotConfig;
}

interface InspectorConfig {
  // What object types this console inspects
  objectTypes: GroveObjectType[];
  // Custom inspector panels per type
  panels?: Record<GroveObjectType, ComponentType>;

  // Default width (can be resized)
  defaultWidth?: number;
}

interface CopilotConfig {
  // Context passed to Copilot
  contextProvider: () => CopilotContext;

  // Available actions for this console
  actions: CopilotAction[];

  // Model preference
  preferredModel?: 'local' | 'hybrid' | 'cloud';
}
```

Why This Pattern?

1. **Consistency**: Users learn once, apply everywhere
2. **Copilot slot**: AI assistance built into every surface
3. **Inspector integration**: Context-aware detail always available
4. **Metrics visibility**: Operational awareness without navigation
5. **DEX compliance**: Structure supports growth without inhibiting it

---

Part 4: Bedrock Information Architecture

Organized by object model, not legacy feature groupings:

```
/bedrock
├── Dashboard (unified status + quick actions)
│
├── Knowledge Garden (Sprout-centric)
```

```
│   ├── Sprout Queue (moderation workflow)
│   ├── Knowledge Sources (document management)
│   ├── Conflict Resolution (contradiction handling)
│   └── Proposals (Architect suggestions)
│
├── Experience Design (DEX objects)
│   ├── Journey Studio (journeys + nodes)
│   ├── Lens Workshop (lenses + personas)
│   └── Moment Editor (engagement triggers)
│
├── Hub Topology (content organization)
│   ├── Hub Map (visual topology)
│   ├── Spoke Manager (connections)
│   └── Entry Points (navigation seeds)
│
├── System Configuration
│   ├── Feature Flags (runtime toggles)
│   ├── System Voice (personality config)
│   └── Media Library (audio + assets)
│
└── Operations
    ├── Health Monitor (system status)
    ├── Engagement Telemetry (user behavior)
    └── Analytics (Genesis metrics)
```

Navigation Tree (Declarative)

```
// src/bedrock/config/navigation.ts

export const bedrockNavigation: NavigationTree = {
  sections: [
    {
      id: 'dashboard',
      label: 'Dashboard',
      icon: 'dashboard',
      path: '/bedrock',
      consoleId: 'dashboard',
    },
    {
      id: 'knowledge',
      label: 'Knowledge Garden',
      icon: 'park',
      children: [
        { id: 'sprouts', label: 'Sprout Queue', consoleId: 'sprout-queue' },
        { id: 'sources', label: 'Sources', consoleId: 'knowledge-sources' },
        { id: 'conflicts', label: 'Conflicts', consoleId: 'conflict-resolution' },
        { id: 'proposals', label: 'Proposals', consoleId: 'proposals' },
      ],
    },
    // ... declaratively defined
  ],
};
```

Part 5: The Shared Component Library

Bedrock uses a strict component vocabulary:

Primitives (Atoms)

Component	Purpose	Used In
`StatCard`	Single metric display	Metrics row
`GlassPanel`	Elevated container	All panels
`ObjectBadge`	Type + status indicator	Lists, grids
`ActionButton`	Primary/secondary actions	Headers, inspectors
`IconBadge`	Icon with background	Navigation, cards

Composites (Molecules)

Component	Purpose	Pattern
`ObjectList`	Filterable object collection	Navigation column
`ObjectGrid`	Visual object display	Content area
`ObjectCard`	Single object summary	Grid children
`CollectionHeader`	Title + count + actions	List/grid headers
`FilterBar`	Multi-facet filtering	Above collections

Structures (Organisms)

Component	Purpose	Pattern
`BedrockLayout`	Three-column + Copilot	Every console
`InspectorPanel`	Right column container	Object detail
`CopilotPanel`	AI assistant interface	Inspector bottom
`MetricsRow`	Horizontal stat display	Console headers
`NavigationTree`	Hierarchical nav	Left column

Part 6: The Copilot Standard

Every console's inspector includes a Copilot. The Copilot is context-aware:

Copilot Context Protocol

```
// src/bedrock/copilot/context.ts

interface CopilotContext {
  // Current console
  consoleId: string;

  // Selected object (if any)
  selectedObject?: GroveObject;

  // Current view state
  viewState: {
    filters: FilterState;
    sortOrder: SortState;
    expandedIds: string[];
  };

  // Related objects for reference resolution
  relatedObjects: {
    hubs: Hub[];
    journeys: Journey[];
    sprouts: Sprout[];
  };

  // Schema for selected object type
  schema?: ObjectSchema;
}
```

Copilot Capabilities by Console

Console	Copilot Can...
Sprout Queue	Extract claims, detect conflicts, suggest categorization
Knowledge Sources	Analyze documents, recommend hubs, identify gaps
Journey Studio	Optimize paths, detect orphans, suggest connections
Lens Workshop	Generate tone guidance, balance emphasis weights
Feature Flags	Analyze impact, suggest rollout strategies
Health Monitor	Diagnose issues, suggest remediation

The Hybrid Model

Copilot uses the **Ratchet-aligned** model selection:

1. **Structured edits** ("set X to Y") → Local 7B (instant, free, private)
2. **Creative generation** ("make it more mysterious") → Local with cloud fallback
3. **Complex reasoning** ("why is this hub underperforming?") → Cloud API

User always sees which model is active. User can always "Enhance with Frontier."

Part 7: The Feature Conveyor Belt

Nearly all features are **already prototyped** in legacy `/foundation`. Bedrock's job is to produce them consistently.

Feature Inventory (From Legacy)

Feature	Legacy Location	Bedrock Console	Priority
Sprout moderation	SproutQueue.tsx	Sprout Queue	P0
Journey editing	NarrativeArchitect.tsx	Journey Studio	P0
Document upload	KnowledgeVault.tsx	Knowledge Sources	P0
Feature flags	RealityTuner.tsx	Feature Flags	P1
Health checks	HealthDashboard.tsx	Health Monitor	P1
Hub routing	RealityTuner.tsx	Hub Topology	P1
Engagement tracking	EngagementBridge.tsx	Engagement Telemetry	P2
Genesis analytics	Genesis.tsx	Analytics	P2
Audio generation	AudioStudio.tsx	Media Library	P3
System voice	RealityTuner.tsx	System Voice	P3

Production Protocol

For each feature:

1. **Audit legacy** → Document what exists, what works, what's broken
2. **Design console** → Apply BedrockLayout, define navigation/content/inspector
3. **Implement primitives** → Build any missing shared components
4. **Wire Copilot** → Define context, actions, model preferences
5. **Test pattern compliance** → Verify against canonical structure
6. **Document** → Update this spec with any learnings

Part 8: Build Sequence

Sprint 0: Infrastructure (Week 1-2)

Goal: Establish the primitives everything else builds on.

Deliverables:

- `BedrockLayout.tsx` — The canonical console structure
- `BedrockNav.tsx` — Declarative navigation tree
- `BedrockInspector.tsx` — Context-aware inspector shell
- `BedrockCopilot.tsx` — Unified AI assistant interface
- `BedrockUIContext.tsx` — State management
- Shared component library (primitives + composites)
- Navigation configuration schema

Acceptance Criteria:

- Empty console renders with correct structure
- Navigation declaratively configured
- Inspector opens/closes with correct context
- Copilot receives context, displays model indicator

Sprint 1: Sprout Queue (Week 3-4)

Goal: The first complete console. Proves the pattern.

Why Sprout Queue First:

- Most critical to knowledge lifecycle thesis
- Clear data model (Sprout lifecycle states)
- Copilot has obvious value (claim extraction, conflict detection)
- High traffic console — validates performance

Deliverables:

- `SproutQueueConsole.tsx` — Full implementation
- `SproutInspector.tsx` — Sprout detail view
- `SproutCopilot.ts` — Sprout-specific actions

- Filtering by stage, hub, date
- Batch operations (approve, reject, archive)

Sprint 2: Knowledge Sources (Week 5-6)

Goal: Document management with Sprout integration.

Deliverables:

- `KnowledgeSourcesConsole.tsx`
- Upload workflow with Mason ingestion trigger
- Source → Sprout relationship display
- Gap analysis Copilot action

Sprint 3: Journey Studio (Week 7-8)

Goal: The DEX object authoring experience.

Deliverables:

- `JourneyStudioConsole.tsx`
- Visual journey builder
- Node connection editor
- Path optimization Copilot action

Sprint 4: Hub Topology (Week 9-10)

Goal: Visual content organization.

Deliverables:

- `HubTopologyConsole.tsx`
- Visual hub map (D3 or similar)
- Spoke management
- Entry point configuration

Sprint 5+: Remaining Consoles

Follow production protocol for:

- Lens Workshop
- Moment Editor
- Feature Flags
- System Voice
- Health Monitor
- Engagement Telemetry
- Analytics
- Media Library

Part 9: Migration & Deprecation

The Strangler Fig Timeline

```
Phase 1: Parallel Operation
├── /foundation continues serving Genesis
├── /bedrock builds toward feature parity
└── No user-facing changes

Phase 2: Feature Flag Transition
├── Feature flag: bedrock_enabled
├── Authorized users can opt-in to /bedrock
├── Feedback collected, bugs fixed
└── /foundation still default

Phase 3: Default Flip
├── /bedrock becomes default
├── /foundation accessible via flag
├── Migration assistance for any custom configs
└── Deprecation warnings on /foundation

Phase 4: Sunset
├── /foundation removed from routing
```

```
  ├── Legacy code archived (not deleted)
  ├── Documentation updated
  └── Bedrock is now just "Foundation"
```

Feature Parity Checklist

Bedrock must implement ALL of these before Phase 3:

- [ ] Sprout moderation workflow
- [ ] Document upload and management
- [ ] Journey/Node editing
- [ ] Lens/Persona configuration
- [ ] Feature flag management
- [ ] Hub routing configuration
- [ ] System voice versioning
- [ ] Health monitoring
- [ ] Engagement telemetry
- [ ] Genesis analytics
- [ ] Audio track management

Part 10: Success Metrics

Pattern Compliance

Metric	Legacy	Target	How We Measure
Console pattern consistency	30%	100%	All use BedrockLayout
Copilot coverage	12.5%	100%	All consoles have Copilot
Shared component usage	~40%	90%+	Component audit
DEX test passage	Unknown	100%	Manual checklist per feature

Operational

Metric	Target	How We Measure
Time to add new object type	< 1 day	Stopwatch
Time to configure new hub	< 5 clicks + Copilot	User testing
Console load time	< 500ms	Performance monitoring
Copilot response time (local)	< 200ms	Latency tracking

Strategic

Metric	Target	How We Measure
Framework extractability	High	Can we lift Bedrock as standalone?
University demo-readiness	Yes	Can we show this to Purdue?
Documentation completeness	100%	Every pattern documented

Appendix A: Document Cross-References

This Document Updates

Document	Section	Change
`GROVE_FOUNDATION_REFACTOR_SPEC.md`	Status	Now "Legacy Reference"
Project README	Architecture	Add Bedrock section

This Document References

Document	Relationship
`The_Trellis_Architecture__First_Order_Directives.md`	Constitutional foundation
`Kinetic_Framework_Strategic_Vision.md`	Strategic context
`copilot-configurator-vision.md`	Copilot implementation guide
`grove-object.ts`	Data model specification

New Documents Needed

Document	Purpose	Owner
BEDROCK_COMPONENT_CATALOG.md	Shared component documentation	Engineering
BEDROCK_COPILOT_ACTIONS.md	Per-console Copilot capabilities	Engineering
BEDROCK_MIGRATION_RUNBOOK.md	Step-by-step migration guide	Engineering

---

Appendix B: The DEX Compliance Checklist

Every Bedrock feature must pass:

Declarative Sovereignty Test

- [ ] Can a non-technical user change this behavior via config file?
- [ ] Is the config file documented with examples?
- [ ] Does changing config require code deployment? (Must be NO)

Capability Agnosticism Test

- [ ] What happens if the model hallucinates?
- [ ] Does schema validation catch invalid outputs?
- [ ] Is there a human checkpoint before mutation?

Provenance Test

- [ ] Can every displayed fact trace to its source?
- [ ] Is modification history preserved?
- [ ] Can we answer "who changed this and when"?

Organic Scalability Test

- [ ] Can new object types use this feature without code changes?
- [ ] Does the feature support serendipitous discovery?
- [ ] Are there artificial limits that will need removal later?

Closing: The Meta-Pattern

> *The Terminal isn't a "feature"—it's proof of the core thesis.*

Bedrock isn't just an admin interface. It's the **meta-demonstration** of Grove's architecture:

- **Declarative configuration** → All console behavior in config
- **Capability agnosticism** → Copilot works with local or cloud models
- **Provenance as infrastructure** → Every change tracked, every fact sourced
- **Organic scalability** → New object types inherit everything automatically

When we demo Bedrock to universities, we're not showing them an admin panel. We're showing them **what exploration architecture looks like when you build it right.**

The features are proven. The patterns are established. Now we produce them correctly.

This is a constitutional document. Changes require Advisory Council review.

Document History

Version	Date	Author	Changes
1.0	2025-12-30	Jim + Claude	Initial specification