INDEX

Part 1: The Core Architecture (System)

- **Doc A: System Architecture & IPC Bridge**
  - *Content:* Electron Main/Renderer process split, IPC channel map (secure communication), File System access patterns, and Security/Context Isolation.
- **Doc B: Database Dictionary (SQLite Schema)**
  - *Content:* The complete SQL schema for `projects`, `media_library`, `nodes`, `canvases`, `transcripts`.
- **Doc C: Ingest & Forensics Protocol**
  - *Content:* The `ffprobe` logic, file sanitization, local asset management, and "Bucket" logic.
- **Doc D: The Proxy & Cache Protocol**
  - **Content:** The "Performance Layer." Defines the background `ffmpeg` worker, the creation of H.264 proxies (for smooth scrubbing), and JSON-based Waveforms (for instant rendering).

Part 2: The Narrative Engine (Physics)

- **Doc E: Fractal Topology & Physics**
  - *Content:* The "Elastic Column" rules, "Snowplow" effect, Container (Act/Scene) logic, and the Coordinate System ($Y=0$ Floor).
- **Doc F: The Two Timecodes & Propagation**
  - *Content:* The math behind Source Time vs. Timeline Time, Drift calculation, and the "Ripple" logic when duration changes.
- **Doc G: Stacking & Track Logic**
  - *Content:* How Vertical Stacking (Satellites) maps to Timeline Tracks (V1, V2, V3) and collision resolution (Vertical Dodge).

Part 3: Interaction & Features

- **Doc H: User Interaction & Input Map**
  - *Content:* Drag & Drop rules (Ghosting), Drop Zones, Shortcuts (I/O), and Canvas/Timeline sync interactions.
- **Doc I: The Source Monitor & Inspector**
  - *Content:* The new Inspector logic, "Karaoke" Transcript mode, Word-Sticky selection, and Trim payloads.
- **Doc J: Multicam & Sub-Node Isolation**
  - *Content:* How we handle multi-angle clips and the "Internal State Map".

Part 4: Output & Project Management

- **Doc K: Project Lifecycle**

    - *Content:* Creation flow, Dashboard, and Global Defaults (House Standards).
- **Doc L: Export Protocol (FCPXML)**
    - *Content:* Converting the Graph into a linear XML for NLEs.

Doc A: System Architecture & IPC Bridge

Doc A: System Architecture & IPC Bridge

Version: 4.0 (Local Electron/SQLite Refactor)

---

1. Architectural Overview

Story Graph (Local) is a desktop-native application built on the **Electron** framework. It strictly adheres to the **Model-View-Controller (MVC)** pattern where:

- **Model (Backend):** The Main Process (Node.js) manages the SQLite Database, File System, and FFprobe child processes.
- **View (Frontend):** The Renderer Process (React/Vite) manages the UI, Canvas Topology, and User Interaction.
- **Controller (Bridge):** The IPC (Inter-Process Communication) layer acts as the secure API gateway between the two.

1.1 Core Technology Stack

- **Runtime:** Electron (Latest Stable)
- **Backend Logic:** Node.js (Main Process)
- **Database:** SQLite3 (`better-sqlite3` for synchronous speed in Main Process)
- **Media Analysis:** FFprobe (Static Binaries bundled with app)
- **Frontend Framework:** React 18 + Vite
- **Canvas Engine:** React Flow (Customized for Physics)
- **Styling:** TailwindCSS + Shadcn/UI

1.2 Security & Isolation

To prevent Remote Code Execution (RCE) and ensure stability:

1. **Context Isolation:** Enabled (`contextIsolation: true`). The Renderer has NO access to Node.js primitives (`fs`, `require`).
2. **Sandbox:** Enabled. Renderer processes are sandboxed.
3. **Preload Scripts:** We use a `preload.ts` file to expose *only* specific API functions (e.g., `window.api.createProject()`) to the global window object.
4. **CSP (Content Security Policy):** Strict CSP headers prevent loading external scripts or unsigned resources.

---

2. The Main Process (Backend)

The Main Process is the "Source of Truth." It is the *only* place where database writes or file system reads occur.

2.1 Directory Structure (App Data)

On first launch, the Main Process ensures the existence of the following local directory structure in the User's `AppData` (Windows) or `Application Support` (Mac) folder:

```
Plaintext

/StoryGraph_Data/
├── projects.db          -- The Master SQLite Database
├── /Project_Images/     -- Cover images for projects
├── /Backups/            -- Auto-generated .db dumps
└── /Logs/               -- Error and transaction logs
```

2.2 Database Initialization

- **Driver:** `better-sqlite3` (chosen for synchronous performance, crucial for drag-and-drop physics calculations).
- **WAL Mode:** Write-Ahead Logging is enabled (`PRAGMA journal_mode = WAL`) to allow concurrent reads/writes and prevent UI blocking.
- **Migration System:** On startup, the app checks `user_version` PRAGMA. If the schema version is lower than the app version, it executes migration scripts (e.g., `ALTER TABLE nodes ADD COLUMN...`).

2.3 Child Processes (FFmpeg/FFprobe)

- **Execution:** FFprobe is spawned using `child_process.execFile`.
- **Queuing:** A `p-queue` (concurrency: 2) manages ingest to prevent freezing the CPU when importing 100 files at once.
- **Pathing:** Static binaries are located in `resources/bin/` to ensure they work in the packaged production build.

---

3. The IPC Bridge (API Specification)

This section defines the **Contract** between Frontend and Backend. The Frontend calls these methods via `window.api.methodName(payload)`.

3.1 System & Window Management

Channel	Method	Payload	Response	Description

window-minimize	invoke	void	void	Minimizes app window.
window-maximize	invoke	void	void	Toggles maximize/restore.
window-close	invoke	void	void	Gracefully closes DB connections, then app.
app-get-path	invoke	type: 'userData'	string	Returns absolute path to App Data.

3.2 Project Management

Channel	Method	Payload	Response	Description
project-create	invoke	ProjectData	{ projectId: UUID }	Creates row in projects table.
project-get-all	invoke	void	Project[]	Returns all projects for Dashboard.
project-get-details	invoke	{ projectId }	Project	Returns full metadata for one project.
project-update	invoke	ProjectData	boolean	Updates mutable fields (name, description).

project-delete	invoke	{ projectId }	boolean	Cascading delete of all related data.

3.3 Media Ingest & Library

Channel	Method	Payload	Response	Description
media-import	invoke	{ filePaths[], projectId }	MediaItem[]	Runs FFprobe, saves to DB, returns items.
media-get-all	invoke	{ projectId }	MediaItem[]	Fetches all assets for the project.
media-relink	invoke	{ mediaId }	boolean	Opens file picker, updates file_path.
bin-create	invoke	{ projectId, name, parentId }	Bin	Creates a new bin.

3.4 Canvas & Topology (The Engine)

Channel	Method	Payload	Response	Description
canvas-create	invoke	{ projectId, settings }	Canvas	Creates new graph workspace.

node-create	invoke	NodeData	Node	Inserts Spine/Sat/Container node.
node-update-pos	invoke	{ id, parentId, drift }	boolean	Updates structure/physics (Drift/Anchor).
node-update-trim	invoke	{ id, inPoint, outPoint }	boolean	Updates Source Monitor trims.
graph-get-state	invoke	{ canvasId }	Node[]	Returns full graph state for rendering.

3.5 Transcripts

Channel	Method	Payload	Response	Description
transcript-import	invoke	{ mediaId, filePath }	boolean	Parses .srt/.vtt, inserts words/segments.
transcript-get	invoke	{ mediaId }	TranscriptData	Returns segments & word-level data.

---

4. Error Handling & Logging Strategy

4.1 Main Process Logging

We use electron-log to write to a daily rotating log file (main.log).

- **Error Level:** Caught exceptions (DB Lock, FFmpeg crash).
- **Info Level:** App lifecycle events (Start, Quit).
- **Debug Level:** IPC payload tracing (Development only).

4.2 Error Propagation

When a backend operation fails (e.g., SQLite constraint violation):

1. The Main Process catches the error.
2. It logs the full stack trace to disk.
3. It throws a standardized IPC Error Object to the Renderer:
4. JavaScript

```
{
  code: "DB_CONSTRAINT_ERROR",
  message: "Cannot delete Project because it is currently open.",
  details: "SQLITE_BUSY..."
}
```

5.
6.
7. The Renderer (`try/catch` block) displays a Toast Notification with the `message`.

---

5. Data Types & Interfaces

To ensure type safety across the bridge, a shared `types.ts` file is symlinked or copied between Main and Renderer.

Example: The Standard Response Wrapper

TypeScript

```
export interface ApiResponse<T> {
  success: boolean;
  data?: T;
  error?: {
    code: string;
    message: string;
  };
}
```

Example: Node Data Structure

TypeScript

```
export type NodeType = 'SPINE' | 'SATELLITE' | 'CONTAINER';

export interface NodeData {
```

```
  id: string;
  canvas_id: string;
  type: NodeType;
  parent_id: string | null;     // Structural Anchor
  container_id: string | null;  // Visual Container
  drift: number;                // Milliseconds
  media_in_point: number;       // Seconds
  media_out_point: number | null;
  // ... other fields matching Schema
}
```

---

6. Build & Distribution

- **Packager:** `electron-builder`.
- **Code Signing:** Required for Mac (Apple Developer ID) and Windows (EV Cert) to prevent "Unknown Publisher" warnings.
- **ASAR:** Enabled. Source code is packed into an archive to obscure logic and speed up `require` calls.
- **Native Modules:** `better-sqlite3` must be rebuilt against the Electron headers during the `postinstall` script phase using `electron-rebuild`.

Doc B: Database Dictionary (SQLite Schema v4

Doc B: Database Dictionary (SQLite Schema v4.5)

Version: 4.5 (Local Story Graph)

---

1. Core Infrastructure Tables

1.1 Table: `projects`

The root container for all creative work. Defines global settings inherited by canvases.

Column Name	Data Type	Constraints	Description
id	TEXT	PRIMARY KEY	Unique UUID.
name	TEXT	NOT NULL	The Project Title.
client_name	TEXT		Optional client metadata.
company_name	TEXT		Optional production company.
description	TEXT		User notes/brief.
image_path	TEXT		Absolute path to local cover image (in /App_Data/Project_Images/).

default_fps	REAL		"House Standard" frame rate (e.g., 23.976).
default_resolution	TEXT		"1920x1080" string.
status	TEXT	DEFAULT 'In Progress'	Enum: In Progress, On Hold, Finished, Archived.
created_at	DATETIME		Timestamp of creation.
last_edited_at	DATETIME		Timestamp of last modification.

1.2 Table: canvases

The workspaces within a project. Canvases are the parents of Nodes.

Column Name	Data Type	Constraints	Description
id	TEXT	PRIMARY KEY	Unique UUID.
project_id	TEXT	NOT NULL, FK	Foreign Key linking to projects.id.
name	TEXT		Canvas Name (e.g., "Director's Cut").

description	TEXT		Canvas specific notes.
fps	REAL		Frame rate used for Timeline calculations. Inherits from Project or Custom.
resolution	TEXT		Resolution string. Inherits from Project or Custom.
created_at	DATETIME		Timestamp of creation.
updated_at	DATETIME		Timestamp of last active use.

1.3 Table: transaction_log Purpose: Enables the Undo/Redo stack. Since we use SQLite, we cannot rely on memory. Every user action records a "Reversible Transaction" here.

Column Name	Data Type	Constraints	Description
id	TEXT	PRIMARY KEY	Unique UUID.
project_id	TEXT	NOT NULL, FK	Links to projects.id.
action_type	TEXT		e.g., 'NODE_MOVE', 'TRIM', 'CREATE'.

undo_payload	JSON		The state BEFORE the change (Inverse).
redo_payload	JSON		The state AFTER the change.
timestamp	DATETIME	DEFAULT CURRENT_TIMESTAMP	

---

2. Media Management Tables

2.1 Table: `media_library`

The registry of all raw assets. Stores forensic metadata extracted by FFprobe.

Column Name	Data Type	Constraints	Description
id	TEXT	PRIMARY KEY	Unique UUID.
project_id	TEXT	NOT NULL, FK	Foreign Key linking to projects.id.
bin_id	TEXT	FK	Foreign Key linking to bins.id. Nullable (Root level).

file_name	TEXT		Original filename (e.g., "DSC001.mov").
clean_name	TEXT		Sanitized display name (e.g., "DSC001").
file_path	TEXT	NOT NULL	Absolute system path to the source file.
media_type	TEXT		Enum: BROLL, DIALOGUE, MUSIC, IMAGE, ANIMATION, MULTICAM.
fps	REAL		Native frame rate.
width	INTEGER		Native width.
height	INTEGER		Native height.
duration_sec	REAL		Total Duration in seconds.
start_tc_string	TEXT		Embedded Source Timecode (e.g., "01:00:00:00").
total_frames	INTEGER		Total frame count.

file_size_bytes	INTEGER		Size on disk in bytes.
date_created	DATETIME		Creation date from file metadata.
metadata_raw	JSON		Full FFprobe JSON dump.
has_transcript	INTEGER	DEFAULT 0	Boolean flag (0=False, 1=True).
proxy_status	TEXT	DEFAULT 'PENDING'	Enum: PENDING, PROCESSING, READY, FAILED.
proxy_path	TEXT		Path to /Cache/Proxies/UUID.mp4.
waveform_status	TEXT	DEFAULT 'PENDING'	Enum: PENDING, READY.
waveform_path	TEXT		Path to /Cache/Waveforms/UUID.json.

2.2 Table: bins

Recursive directory structure for organizing assets.

Column Name	Data Type	Constraints	Description
id	TEXT	PRIMARY KEY	Unique UUID.
project_id	TEXT	NOT NULL, FK	Foreign Key linking to projects.id.
parent_bin_id	TEXT	FK	Foreign Key linking to bins.id. Nullable (allows nesting).
name	TEXT		Name of the Bin/Folder.

---

3. The Narrative Graph (Topology)

3.1 Table: nodes

The unified table for all topological elements. This powers the Physics Engine and Timeline.

Column Name	Data Type	Constraints	Description
id	TEXT	PRIMARY KEY	Unique UUID.
canvas_id	TEXT	NOT NULL, FK	Foreign Key linking to canvases.id.

type	TEXT	NOT NULL	Enum: SPINE, SATELLITE, CONTAINER.
container_role	TEXT		Enum: ACT, SCENE. Null if type is not Container.
asset_id	TEXT	FK	Foreign Key linking to media_library.id. Null for Containers.
parent_id	TEXT	FK	**Structural Anchor.** UUID of the node this is attached to.
anchor_type	TEXT		Enum: APPEND, PREPEND, TOP, ORIGIN.
container_id	TEXT	FK	**Visual Wrapper.** UUID of the Act/Scene this sits inside.
drift	INTEGER	DEFAULT 0	Milliseconds offset from Anchor (+ Gap, - Overlap).
ui_track_lane	INTEGER		Vertical Sort Hint (V1, V2...) for Timeline rendering.

media_in_point	REAL	DEFAULT 0	**Source Monitor Trim.** Start of cut relative to source (Seconds).
media_out_point	REAL		**Source Monitor Trim.** End of cut relative to source (Seconds). Null = End of File.
playback_rate	REAL	DEFAULT 1.0	**1.0 = Normal, 0.5 = Slow Mo, 2.0 = Fast Fwd.**
active_channels	JSON		**Array of active audio channels (e.g., [1, 2]).**
transition_in_type	TEXT		**Enum: 'CUT', 'DISSOLVE', 'WIPE'.**
transition_in_duration	INTEGER	DEFAULT 0	**Duration of the overlap in Frames.**
internal_state_map	JSON		**For Multicam: Stores the edit decisions (cuts).**

---

4. Semantic Data (Transcripts)

4.1 Table: `transcript_segments`

High-level text blocks grouped by speaker turn. Used for Inspector display.

Column Name	Data Type	Constraints	Description
id	TEXT	PRIMARY KEY	Unique UUID.
media_id	TEXT	NOT NULL, FK	Foreign Key linking to media_library.id.
speaker_name	TEXT		Speaker label (e.g., "Interviewer").
time_in_sec	REAL		Start time of the first word in the segment.
time_out_sec	REAL		End time of the last word in the segment.
content_blob	TEXT		Full text string of the segment for search/display.
segment_order	INTEGER		Sequence index (1, 2, 3...) to maintain flow.

4.2 Table: `transcript_words`

Atomic word data. Used for "Word Sticky" editing and precision trimming.

Column Name	Data Type	Constraints	Description
id	TEXT	PRIMARY KEY	Unique UUID.
segment_id	TEXT	NOT NULL, FK	Foreign Key linking to transcript_segments.id.
word	TEXT		The actual word string.
time_in_sec	REAL		Exact start time of the word.
time_out_sec	REAL		Exact end time of the word.

---

5. Advanced Features (Multicam)

5.1 Table: multicam_members

Defines the internal sync map for a Multicam Asset.

Column Name	Data Type	Constraints	Description
id	TEXT	PRIMARY KEY	Unique UUID.
multicam_asset_id	TEXT	NOT NULL, FK	FK to media_library (The Container Asset).

angle_asset_id	TEXT	NOT NULL, FK	FK to media_library (The ISO File).
angle_name	TEXT		Name of the angle (e.g., "Cam A", "Wide").
sync_offset_frames	INTEGER	DEFAULT 0	Frame offset relative to Container Start.

Doc C: Ingest & Forensics Protocol

Doc C: Ingest & Forensics Protocol

Version: 2.0 (Local FFprobe Integration)

---

1. Protocol Overview

This protocol defines the strict lifecycle of a media asset entering the Story Graph ecosystem. Unlike traditional NLEs that may lazily load metadata, Story Graph performs a **Forensic Scan** immediately upon ingest. This ensures that every asset in the `media_library` has a guaranteed set of technical data (FPS, Duration, TC) before it can be used in the graph.

1.1 Core Principles

1. **Zero-Trust Ingest:** We do not assume a file is valid just because it has a `.mov` extension. We verify it with `ffprobe`.
2. **Immutable Source:** We never modify the user's source files. We only read them.
3. **Sanitized Identity:** Every file gets a `clean_name` and a UUID (`id`) to decouple the internal logic from the file system filename.

---

2. The Ingest Pipeline

Step 1: The Trigger

User Action:

- Drag & Drop files/folders onto the "Media Library" panel.
- Click "Import Media" button $\to$ System File Picker.

Frontend Payload:

The Renderer sends an IPC message to the Main Process:

```
TypeScript

ipcRenderer.invoke('media-import', {
  projectId: "UUID...",
  filePaths: ["/Users/Me/Footage/A001.mov", "/Users/Me/Audio/Music.mp3"]
});
```

Step 2: The Queue (Main Process)

To prevent UI freezing or CPU spikes when importing hundreds of files, we use a concurrency queue.

- **Concurrency Limit:** 2 active FFprobe processes max.
- **Batching:** The UI shows a global progress bar (e.g., "Importing 1 of 50...").

Step 3: Forensic Scanning (FFprobe)

For each file path, the Main Process spawns a child process:

ffprobe -v quiet -print_format json -show_format -show_streams "FILE_PATH"

FFprobe Field	Target DB Column	Logic / Fallback
format.filename	file_path	Absolute path.
format.tags.creation_time	date_created	If missing, use OS fs.stat.birthtime.
format.duration	duration_sec	Parse string to float.
streams[v].width	width	Video stream only.
streams[v].height	height	Video stream only.
streams[v].r_frame_rate	fps	Calculate (e.g., "24000/1001" -> 23.976).
streams[v].nb_frames	total_frames	If missing, calc duration * fps.

format.tags.timecode	start_tc_string	If missing, default to "00:00:00:00".

Step 4: Auto-Classification

The system automatically assigns a media_type based on the stream data:

1. **Has Video Stream?** $\to$ 'BROLL' (Default).
2. **Audio Stream Only?** $\to$ 'MUSIC'.
3. **Image Format (png/jpg)?** $\to$ 'IMAGE'.
4. **XML/FCXML Extension?** $\to$ Trigger **Multicam Protocol** (See Section 4).

Step 5: Sanitization

We generate a clean_name to be used in the UI.

- **Rule:** Remove file extension. Remove common camera suffixes if config enabled (e.g., _PROXY).
- **Example:** Interview_CamA_001.mov $\to$ Interview_CamA_001.

Step 6: Persistence

The sanitized data is inserted into the media_library table.

- bin_id: Defaults to NULL (Root Level) or the active Bin ID if user dropped *into* a specific bin.

---

3. Local Asset Management

3.1 File Tracking

The application stores **Absolute Paths**.

- **Risk:** User moves file A001.mov from Downloads to External Drive.
- **Detection:** On App Launch (or when loading a Project), the system runs a fast fs.access() check on all file_path entries.
- **State:**
    - **Found:** Normal behavior.
    - **Missing:** UI shows Red "!" Badge. Thumbnail is greyed out.

3.2 Relinking Workflow

User Action: Click "Relink" on a missing asset.

1. System opens File Picker.
2. User selects the new location of the file.
3. **Verification:** System runs a quick FFprobe on the *new* file.
    - **Check:** Does `total_frames` and `streams` match the DB record?
    - **Match:** Update `file_path` in DB. Remove "Missing" badge.
    - **Mismatch:** Show Warning: "This file appears to be different (Duration mismatch). Link anyway?"

---

4. The "Bucket" Logic (Staging Area)

The **Bucket** is a unique zone in Story Graph. It is a "Parking Lot" for nodes that have been created (ingested/trimmed) but not yet placed in the narrative topology.

4.1 Definition

- **Technical:** A `SPINE` or `SATELLITE` node where `parent_id` is `NULL` and `canvas_id` matches the current canvas, but it is **not the Origin**.
- **Storage:** These nodes exist in the `nodes` table just like any other node.
- **Visuals:** They are rendered in a separate sidebar panel, not on the main topological infinite canvas.

4.2 Ingest to Bucket

User Action:

1. Open Inspector (Source Monitor).
2. Trim a range (`in` / `out`).
3. Click "Add to Bucket" (instead of dragging to Canvas).

Backend Action:

- Create Node in `nodes` table.
- `type`: `'SPINE'` (usually).
- `parent_id`: `NULL`.
- `media_in_point` / `media_out_point`: Saved.
- `container_id`: `NULL`.

4.3 Drag from Bucket

User Action: Drag a node from the Bucket sidebar onto the Canvas.

- **Backend:**
    - Update `parent_id` $\to$ Target Anchor Node UUID.

    - Update container_id $\to$ Target Container UUID.
- **Result:** The node "leaves" the Bucket list and physically appears on the Graph.

---

5. Special Case: Batch Transcript Sync

5.1 The Handshake

If the user imports a video file (Interview.mp4) and a transcript file (Interview.srt) **at the same time**:

1. The Ingest Engine notices the matching filenames (ignoring extension).
2. It processes the Video first $\to$ Gets media_id.
3. It automatically triggers transcript-import using that media_id.
4. **Result:** The video arrives in the bin with the "Transcript Attached" badge already active.

Doc D: The Proxy & Cache Protocol

Doc D: The Proxy & Cache Protocol

1. Executive Summary

Objective: Eliminate playback latency and UI blocking during the editing of high-bitrate media (4K/8K, ProRes RAW, H.265).

The Constraint: Electron's Renderer (Chromium) has limited codec support and single-threaded UI rendering. It cannot natively decode professional acquisition formats efficiently.

The Solution: Implement a "Shadow File System". For every media asset ingested, the system asynchronously generates:

1. **Visual Proxy:** A lightweight, hardware-accelerated H.264 file.
2. **Data Waveform:** A JSON-based array of audio peaks for immediate graph rendering.

2. Directory Architecture (The Cache)

The application maintains a strict folder hierarchy within the User's Application Data folder (OS-dependent). This is **not** inside the Project Folder (to prevent bloating portable project files), but cache artifacts are linked via unique Asset IDs.

Path Structure:

```
Plaintext

/User_Library/Application_Support/StoryGraph/Cache/
├── Proxies/
│   └── [Asset_UUID]_proxy.mp4
├── Waveforms/
│   └── [Asset_UUID]_wave.json
└── Thumbnails/
    └── [Asset_UUID]_thumb.jpg
```

3. The Render Worker (Process Logic)

Architecture: The Main Process spawns a hidden, detached **Node.js Child Process** (The "Ingest Worker") to handle FFmpeg operations. This ensures the Main Thread (IPC) and Renderer Thread (React) never stutter during ingest.

Queue Logic:

- **FIFO (First-In-First-Out):** Default behavior.
- **Priority Interrupt:** If a user drags a "Pending" clip onto a Canvas, that Asset ID is bumped to the front of the queue.

4. Technical Specifications

4.1 Visual Proxy Spec

Targeting maximum compatibility with Chromium's HTML5 `<video>` tag.

- **Container:** `.mp4`
- **Video Codec:** H.264 (libx264)
- **Profile:** Baseline (Low complexity for fast seek).
- **Resolution:** Fixed Height 720p (Maintain Aspect Ratio).
- **Bitrate:** Constant Rate Factor (CRF) 23 (Balance between visual quality and decode speed).
- **Audio Codec:** AAC (128k stereo).
- **FFmpeg Construction:**
- Bash

```
ffmpeg -i "Input.mov" -vf "scale=-2:720" -c:v libx264 -profile:v baseline -level 3.0 -preset ultrafast -crf 23 -c:a aac -b:a 128k -movflags +faststart "Output_proxy.mp4"
```

-
-

4.2 Waveform Spec

We do not generate images (PNGs) as they pixelate during Timeline Zoom. We generate **Data**.

- **Format:** JSON Array of Floating Points (0.0 to 1.0).
- **Resolution:** 100 samples per second of video.
- **Process:**
    1. FFmpeg extracts raw PCM data.
    2. Worker calculates Root Mean Square (RMS) amplitude for every 1/100th second chunk.
    3. Writes `[0.01, 0.4, 0.6, ...]` to `_wave.json`.
- **React Implementation:** The frontend reads this JSON and draws the waveform on an HTML5 Canvas element. This allows infinite smooth zooming without regeneration.

5. Database Schema Integration

This protocol requires specific columns in the `media_library` table (defined in Doc B) to track state.

Status Enum (`proxy_status`):

- `PENDING`: In queue.
- `PROCESSING`: Currently being transcoded.
- `READY`: Available on disk.
- `FAILED`: FFmpeg error (User must retry).
- `OFFLINE`: Source file missing, proxy unavailable.

6. Failover & Behavior Protocols

Protocol A: The "Switch-on-Availability"

The React Video Player component has a src prop logic:

```
TypeScript

const videoSource = (file.proxy_status === 'READY')
    ? "file://" + file.proxy_path
    : "file://" + file.file_path; // Fallback to source (may lag)
```

Protocol B: The "Ghost Waveform"

While waveform_status is PENDING, the timeline clip renders a flat line or a "Generating..." loader pattern. It does not attempt to read the source audio file for visualization (too expensive).

Protocol C: Cache Cleaning

On Application Exit, the system checks the last_accessed date of Cache files.

- **Rule:** Delete proxies older than 30 days.
- **Logic:** Proxies are disposable. Source media is sacred.

Doc E: Fractal Topology & Physics

Doc E: Fractal Topology & Physics

Version: 1.5 (Local Physics Engine)

---

1. The Coordinate System

The Story Graph Canvas is an infinite 2D plane where structure determines position. Unlike a timeline (where X = Time), here **X = Sequence** and **Y = Hierarchy**.

1.1 The Floor (`Y = 0`)

- **Definition:** The absolute horizontal axis where the main narrative spine lives.
- **Rules:**
    - All **Act Containers** sit on `Y=0`.
    - All **Scene Containers** inside Acts sit on `Y=0` (relative to the Act).
    - All **Spine Nodes** (Dialogue/Audio) inside Scenes sit on `Y=0` (relative to the Scene).
    - Nothing can go "below" the floor (Positive Y is reserved for future features or restricted).

1.2 The Sky (`Y < 0`)

- **Definition:** The vertical space above the spine, used for visual context (B-Roll, Images).
- **Rules:**
    - **Negative Y** represents "Up".
    - **Stacking:** Satellite nodes stack upwards. A node at `Y = -450` is visually *above* a node at `Y = -150`.

1.3 The Flow (X-Axis)

- **Definition:** Time flows Left to Right.
- **Grid:** 1 unit = 1 pixel (internal logic).
- **Gaps:** Standard gap between nodes is `100px`.

---

2. The "Elastic Column" Physics

This is the most critical algorithm in the application. It ensures that the visual structure always accommodates the content, preventing overlaps without manual resizing.

2.1 The Principle

Every Spine Node (and every Container) defines a vertical "Column" of territory. The width of this column is **dynamic**.

2.2 Calculation Logic

For any given Spine Node (`S`):

1. **Base Width:** Fixed at `300px`.
2. **Tree Scan:** The engine scans all Satellite nodes attached to `S`.
3. **Wing Span:** It identifies the *Left-Most Edge* and *Right-Most Edge* of the satellite tree (e.g., a "Left Wing" satellite might extend -300px to the left).
4. Effective Width:
   ColumnWidth = MAX(300, (RightWingEdge - LeftWingEdge) + Padding)

2.3 The "Snowplow" Effect

If the Effective Width of Node A increases (e.g., user drops a wide satellite structure on it):

1. The system calculates the `Delta` (NewWidth - OldWidth).
2. **Action:** All subsequent siblings (Node B, C, D...) are mechanically pushed to the right by `Delta`.
3. **Recursion:** If Node A is inside "Scene 1", "Scene 1" also grows wider. This pushes "Scene 2" to the right.

---

3. Fractal Containers (Acts & Scenes)

Containers are not just background images; they are physical nodes that participate in the Snowplow logic.

3.1 Hierarchy Levels

1. **Level 1: The Canvas** (Root).
2. **Level 2: Act Nodes** (Container). Anchored to the Origin or Previous Act.
3. **Level 3: Scene Nodes** (Container). Anchored inside an Act.
4. **Level 4: Spine Nodes** (Media). Anchored inside a Scene.
5. **Level 5: Satellite Nodes** (Media). Anchored to a Spine Node.

3.2 Container Sizing (Auto-Fit)

A Container (Act/Scene) has no fixed width. It is **Elastic**.

- Formula:
  ContainerWidth = SUM(ChildWidths) + (ChildCount * Gap) + InternalPadding
- **Interaction:** You cannot manually resize an Act. You resize it by dropping more content inside it.

---

4. Anchoring Rules (The Rigid Structure)

Every node must have a specific valid connection to exist on the graph.

4.1 Valid Anchor Types

Anchor Type	Target	Description	Physics Result
`ORIGIN`	`NULL`	First item on Canvas.	`X=0, Y=0`
`APPEND`	`Node`	Attached to Right Edge.	`X = Parent.X + Parent.Width + Gap`
`PREPEND`	`Node`	Attached to Left Edge (Satellites only).	`X = Parent.X - Self.Width - Gap`
`TOP`	`Node`	Attached to Top Edge (Satellites only).	`Y = Parent.Y - Self.Height - Gap`

4.2 The "Ghost Box" Preview

When a user drags a node over the canvas:

1. The system raycasts to find the nearest valid Anchor Point.
2. It calculates the **Hypothetical Physics State** (e.g., "If dropped here, the column grows by 400px").
3. **Visual:** A semi-transparent "Ghost" box appears, and the rest of the graph "Snowplows" out of the way *before* the user releases the mouse.

---

5. Universal Drift (Temporal Offset)

While the Canvas coordinates determine *Sequence*, the **Drift** property determines *Timing*.

- **Definition:** `Drift` is the time offset (in milliseconds) between a Node and its Anchor.
- **Physics Independence:** Changing `Drift` (e.g., creating a gap) **NEVER** moves the node on the Canvas.
- **Visual Feedback:**

- **0ms:** Solid Connection Line.
- **>0ms:** Dashed Line + Label [+5s].
- **<0ms:** ZigZag Line + Label [-2s].

Doc F: The Two Timecodes & Propagation

Doc F: The Two Timecodes & Propagation

Version: 1.0 (Canvas-Timeline Calculus)

---

1. Core Concept: The Separation of Times

In standard NLEs, a clip has a fixed position on a timeline. In Story Graph, a Node's position is **Derived**. We separate the *Internal* time of the media from the *External* time of the narrative.

1.1 Source Time (Internal)

- **Definition:** The specific range of the raw media file that is visible.
- **Reference:** The Source File's embedded timecode (e.g., 01:00:00:00).
- **Storage:** Stored in the nodes table as media_in_point and media_out_point (in seconds relative to the file start).
- **Formula:**

$$Duration = \frac{\{media\_out\_point - media\_in\_point\}\{playback\_rate\}}{\{playback\_rate\}}$$

    - Note: If playback_rate is 0.5 (50% speed), the Timeline Duration doubles.

1.2 Timeline Time (External)

- **Definition:** The calculated absolute position of the node on the Temporal Floor.
- **Reference:** The Canvas Origin (00:00:00:00).
- **Storage: NONE.** This is never stored. It is calculated live on every render frame.
- **Formula:** Timeline_In = Anchor_Node.Timeline_Out + Drift.

---

2. The Calculation Algorithm

The "Temporal Floor" is a read-only view generated by traversing the graph.

2.1 The Traversal Loop

The engine performs a Depth-First Traversal of the Node Graph to calculate times.

Step 1: Find the Origin

- Identify the Node where anchor_type = 'ORIGIN'.
- Origin.Timeline_In = 0.
- Origin.Timeline_Out = Origin.Duration.

Step 2: Recursive Chain

For every Child Node attached to a Parent:

1. **Get Anchor Time:**
    - If `anchor_type = 'APPEND'`, Anchor Time = `Parent.Timeline_Out`.
    - If `anchor_type = 'PREPEND'`, Anchor Time = `Parent.Timeline_In`.
    - If `anchor_type = 'TOP'`, Anchor Time = `Parent.Timeline_In`.
2. **Apply Drift & Transition:**

$$Child.Timeline\_In = Anchor\_Time + Child.Drift - Child.Transition\_In\_Duration$$

    - (Note: If a 24-frame dissolve exists, the clip effectively starts earlier to overlap the outgoing clip.)
3. **Calculate Duration:**

$$Child.Duration = \frac{Child.Media\_Out - Child.Media\_In\}\{Child.Playback\_Rate\}}{\{Child.playback\_rate\}}$$

4. **Calculate End:**

$$Child.Timeline\_Out = Child.Timeline\_In + Child.Duration$$

---

3. Propagation Logic (The Ripple)

When a user edits a node, the change propagates downstream. This is the "Ripple" effect.

3.1 Scenario: Duration Change

User extends **Node A** by trimming the `out_point` by +2 seconds.

- **Before:**
    - Node A Duration: 5s. (Ends at 00:05).
    - Node B (Attached to A, Drift 0): Starts at 00:05.
- **Action:**
    - User drags Node A handle in Inspector. `nodes.media_out_point` updates.
    - Node A Duration becomes 7s.
- **The Math:**
    - Node A `Timeline_Out` is recalculated $\to$ 00:07.
    - Node B checks its anchor (Node A).
    - `Node B Start = Node A End (00:07) + Drift (0)`.
    - Node B `Timeline_In` becomes 00:07.
- **Visual Result:**
    - On Canvas: Node A stays the same width (fixed topology).
    - On Timeline: Clip A grows longer. Clip B slides to the right.

3.2 Scenario: Drift Adjustment

User drags **Clip B** to the right on the Timeline to create a gap.

- **Action:** User moves Clip B from 00:07 to 00:09.
- **The Math:**
  - System calculates `Delta = New_Start (9) - Anchor_End (7)`.
  - `Drift = +2 seconds`.
  - Update `nodes.drift` = `2000` (ms).
- **Visual Result:**
  - On Timeline: A gap appears.
  - On Canvas: Nodes do not move. The connector line becomes dashed with label `[+2s]`.

3.3 Scenario: Variable Speed (Retiming)

- **Context:** Node A (Spine) is followed by Node B.
- **Action:** User changes Node A playback_rate from 1.0 to 0.5 (Slow Motion).
- **Physics:**
  1. Node A media_points remain unchanged (Source Integrity).
  2. Node A Timeline_Duration doubles (e.g., 2s $\to$ 4s).
  3. Node A Timeline_End pushes out by +2 seconds.
     4. Ripple: Node B (anchored to A) reads the new End Time and slides +2s to the right1.
  4. **Visual:** The Canvas width of Node A expands. The Connector Line to Node B remains solid (Drift = 0).

---

4. Paradox Prevention

Because time is relative, we must prevent circular logic.

4.1 The Loop Check

- **Rule:** A Node cannot be an ancestor of its own anchor.
- **Enforcement:** Before updating `parent_id`, the system runs a recursive check up the tree.
- **Error:** If `New_Parent_ID` is found in `Self.Children`, the drag is rejected, and the Node snaps back to its original position.

Doc G: Stacking & Track Logic

Doc G: Stacking & Track Logic

Version: 1.0 (Auto-Layout Engine)

---

1. The Conceptual Bridge

In the Story Graph, "Tracks" are a calculated resource, not a rigid container.

- **On Canvas:** Relationships are topological (Parent/Child). A node sits "Above" another node.
- **On Timeline:** Relationships are layered (V1, V2, V3).
- **The Goal:** The system must automatically map the Topology to Track Lanes so the user never has to manually "patch" tracks.

---

2. Track V1: The Narrative Spine

The "Floor" of the Canvas ($Y=0$) maps exclusively to **Track V1**.

2.1 Rules of V1

- **Exclusivity:** Only nodes with `type='SPINE'` or `type='CONTAINER'` exist on V1.
- **No Overlap:** The "Snowplow" physics (Doc D) guarantees that no two Spine nodes ever occupy the same X-coordinates. Therefore, V1 never has collisions.
- **Gaps:** Gaps on V1 represent silence or black video.

---

3. The Satellite Algorithm (V2+)

All nodes with `type='SATELLITE'` populate the tracks above V1. Since satellites can drift and overlap in time, the system runs a **"Tetris Algorithm"** to assign tracks dynamically.

3.1 The Auto-Layout Pass

Every time the Graph recalculates (e.g., after a drag event), the render engine executes:

1. **Fetch:** Get all Satellite nodes active in the current view.
2. **Sort:** Order them by `Timeline_In` (Start Time).
3. **Allocate:**
    - Initialize `Tracks = [V2, V3, V4...]`.
    - For each Satellite `S`:
        - Check **V2**: Is the space `S.Start` to `S.End` empty?
            - **Yes:** Assign `S` to V2.
            - **No:** Check **V3**.
            - Repeat until a free lane is found.

3.2 The `ui_track_lane` Override

The algorithm above is purely mathematical. However, users often want specific vertical ordering (e.g., "Graphics always on V4").

- **Database Field:** `nodes.ui_track_lane` (Integer).
- **Logic:**
    - If `S.ui_track_lane` is set (e.g., `3`), the algorithm tries to place `S` on **V3** first.
    - **Collision Fallback:** If V3 is occupied by another locked node, it bumps `S` to **V4**.

---

4. Collision Resolution (The "Vertical Dodge")

This logic handles what happens when a user drags a clip *on the Timeline* and causes a collision.

4.1 Scenario

- **State:** Clip A is on V2. Clip B is on V2. They are far apart.
- **Action:** User drags Clip B to the left. It now overlaps Clip A.
- **NLE Behavior (Old):** Block the move or overwrite Clip A.
- **Story Graph Behavior (New):**
    1. System detects intersection on V2.
    2. System automatically promotes Clip B to **V3**.
    3. **Visual:** Clip B "hops" up a track visually as it slides over Clip A.
    4. **Data:** `nodes.ui_track_lane` for Clip B is updated to `3`.

---

5. Special Topologies

5.1 The Stack (Composites)

- **Canvas Action:** User drops Node B onto the **Top Edge** of Node A.
- **Canvas Visual:** Node B sits physically on top of Node A ($Y = Y_{parent} - Height$).
- **Timeline Result:**
    - Node A is on **V(n)**.
    - Node B is strictly assigned to **V(n+1)**.
    - They play simultaneously (Composite/PIP).

5.2 The Wings (J-Cuts / L-Cuts)

- **Canvas Action:** User drops Node B on the **Left/Right Edge** of Node A.
- **Timeline Result:**

- Node A is on **V1** (Spine).
- Node B is on **V2** (Context).
- The Overlap (J/L cut portion) is handled naturally by the tracks.

Doc H: User Interaction & Input Map

Doc H: User Interaction & Input Map

Version: 2.0 (Bi-Modal Graph/Timeline)

---

1. The Interaction Philosophy

Story Graph operates in two simultaneous modes. The user input context determines whether they are modifying **Structure** (Canvas) or **Timing** (Timeline).

1.1 Bi-Directional Sync (Highlighting)

- **Rule:** The Node and the Clip are the same object (`id`).
- **Behavior:**
    - **Hover Canvas Node:** The corresponding Clip on the Timeline highlights (White Border).
    - **Hover Timeline Clip:** The corresponding Node on the Canvas highlights.
    - **Select:** Clicking either selects *both*. The Inspector populates immediately.

Interaction Context	User Action	System Response
Hover Canvas Node	Mouse over Node Body	Corresponding Timeline Clip highlights (White Border).
Hover Timeline Clip	Mouse over Clip Body	Corresponding Canvas Node highlights (White Border).
Selection	Click either Object	Both are selected. Inspector populates with `id` data.

1.2 Focus State

- **Canvas Focus:** Click anywhere on the background or a node. Keyboard shortcuts apply to Graph Topology (e.g., Delete Node).
- **Timeline Focus:** Click the Timeline header or a track. Keyboard shortcuts apply to Playhead/Drift (e.g., Split Clip).

Focus Zone	Trigger Action	Keyboard Scope	Example Shortcut
Canvas Focus	Click Background or Node	Graph Topology	`Delete` (Removes Node & Children)
Timeline Focus	Click Header or Track	Playhead / Drift	`Cmd+K` (Split Clip)

---

2. Mouse & Trackpad Protocol

2.1 Navigation (Pan & Zoom)

Standardized controls across both Canvas and Timeline.

Action	Input Method	Result
Pan	Middle Mouse Drag OR Space + Left Drag	Moves the viewport (X / Y coordinates).
Zoom	Mouse Wheel OR Trackpad Pinch	Scales the view around the cursor pointer.

Horizontal Scroll	Shift + Mouse Wheel	Scrolls Timeline/Canvas horizontally (Time/Sequence).
Vertical Scroll	Mouse Wheel	Scrolls vertically (Stacking / Tracks).

2.2 Selection Logic

Action	Input	Result
Single Select	Left Click	Selects object. Deselects others. Loads Inspector.
Add Select	Shift + Click	Adds object to selection array.
Range Select	Drag on Background	Draws Marquee Box. Selects all *fully* contained items.
Container Select	Double Click (Container)	"Dives In" (Focus enters Scene/Act).

---

3. Drag & Drop Physics (The Core)

This is the primary method for building the narrative.

3.1 The Drop Zones (Canvas)

When dragging a payload (Media Item or existing Node) over a Target Node, invisible zones trigger different Anchors.

Drop Zone on Target	Anchor Type	Visual Feedback	Resulting Topology
Right Edge (50%)	APPEND	Ghost Box snaps to Right.	New Node attaches to end.
Left Edge (20%)	PREPEND	Ghost Box snaps to Left.	New Node attaches to start.
Top Edge (30%)	TOP	Ghost Box snaps to Top.	New Node stacks above (Composite).
Center (Container)	INSIDE	Container Border glows.	New Node appended to Container content.

3.2 The Ghosting System

- **State:** User is dragging, mouse is down.
- **Calculation:** Every 16ms (60fps), the engine Raycasts to find the nearest valid Drop Zone.
- **Visual:**
    1. **Ghost Box:** Semi-transparent replica of the node appears at the snap position.
    2. **Snowplow Preview:** The existing graph *actually moves* (animates) to show the post-drop layout.
    3. **Validation:** If the move creates a Paradox (Loop), the Ghost turns **Red**.

---

4. Timeline-Specific Interactions

Interactions here modify `drift` and `trim`, never `parent_id`.

4.1 Clip Manipulation

Action	Input	Backend Update	Visual Result
Slide (Drift)	Drag Clip Body	`nodes.drift`	Gap opens/closes. Connector dashed/solid.
Trim Head	Drag Left Edge	`media_in_point`	Clip Start moves. *Ripple:* Duration changes.
Trim Tail	Drag Right Edge	`media_out_point`	Clip End moves. *Ripple:* Next clip slides.
Slip	Alt + Drag Body	`media_in` & `media_out`	Content slides *inside* the clip. Duration fixed.
Vertical Move	Drag Up/Down	`ui_track_lane`	Clip hops tracks (V1 -> V2).

---

5. Keyboard Shortcut Map

5.1 Global Shortcuts

Key	Function	Scope
Space	Play / Pause	Global
Cmd + Z	Undo	Global (Transaction Rollback)
Cmd + S	Save Project	Global (Force SQLite Commit)
Delete / Bksp	Delete Selection	Removes Node & Children (Cascade Warning).
F	Frame Selection	Zooms Viewport to fit selection.

5.2 Editing Shortcuts (Inspector/Canvas)

Key	Function	Logic
J / K / L	Rev / Stop / Fwd	Variable speed playback (1x, 2x, 4x).
I	Mark In	Sets media_in_point to Playhead.
O	Mark Out	Sets media_out_point to Playhead.
M	Add Marker	Adds metadata marker at Playhead.
~ (Tilde)	Maximize Panel	Maximizes the hovered panel (Canvas/Timeline).

5.3 Multicam (Isolation Mode)

Only active when a Multicam Node is selected.

Key	Function	Logic
`1` - `9`	Cut to Angle	Updates `internal_state_map` at current Playhead time.
`0`	Flatten	Reverts to container view.

Doc I: The Source Monitor & Inspector

Doc I: The Source Monitor & Inspector

Version: 2.0 (Integrated Transcript Editor)

---

1. Panel Overview

The Inspector is a context-aware panel that changes its function based on the user's selection.

- **State A: Source Monitor:** When a raw asset is selected in the *Media Library*. (Read-Only Source).
- **State B: Node Editor:** When an existing Node is selected on the *Canvas*. (Live Read/Write).

1.1 UI Components

The layout remains consistent across both states to preserve muscle memory.

Component	Function	visual_state
Header	Displays `clean_name` and resolution/FPS badge.	Static Text
Preview Player	HTML5 Video or Waveform Canvas.	Draggable Surface (Source Drag).
Mini-Timeline	Scrubber, Playhead, and Range Bar (In/Out).	Range Bar Color: **Blue** (Source), **Green** (Node).
Transcript View	Scrollable text block grouped by Speaker.	Highlighted "Karaoke" Word.

---

2. The "Karaoke" Engine (Playback)

This system ensures the transcript visually follows the video playback in real-time.

2.1 Synchronization Logic

- **Trigger:** `requestAnimationFrame` loop (active only when Inspector is visible + playing).
- **Query:** Compare `video.currentTime` against the `transcript_words` array loaded in memory.
- **Condition:** `IF (currentTime >= word.time_in AND currentTime <= word.time_out)`
- **Action:** Apply CSS class `.active-word` (Yellow Highlight) to the specific DOM element.

2.2 Auto-Scroll

- **Behavior:** If the active word is outside the visible viewport of the Transcript container.
- **Action:** `container.scrollTop = activeWordElement.offsetTop - (containerHeight / 2)`.
- **Result:** The active line stays vertically centered.

---

3. Word-Sticky Selection (Semantic Editing)

This feature allows users to edit based on *what is said* rather than looking at waveforms.

3.1 Selection Modes

Mode	User Action	Logic
Click-to-Seek	Click any word.	Player jumps to `word.time_in`.
Mark In	Press `I` (or click btn).	`in_point` snaps to **Start** of current/selected word.
Mark Out	Press `O` (or click btn).	`out_point` snaps to **End** of current/selected word.
Drag Text	Click & Drag cursor over text.	**Mouse Down:** Set In (`word[0].start`).

		Mouse Up: Set Out (`word[n].end`).

3.2 The Feedback Loop

- **Text $\to$ Timeline:** Selecting text updates the visual Range Bar on the Mini-Timeline.
- **Timeline $\to$ Text:** Dragging the Mini-Timeline handles manually will deselect the specific words (visual dimming) but keep the time range active. This allows for sub-word breath removal.

---

4. The Drag Payload (Source Monitor)

When dragging the video image *from the Inspector* to the *Canvas*, a JSON payload is constructed.

4.1 Payload Structure

```JSON
{
  "type": "MEDIA_DROP",
  "payload": {
    "asset_id": "550e8400-e29b-...",
    "media_in_point": 14.532,
    "media_out_point": 18.205,
    "duration": 3.673,
    "ghost_width": 300 // Base width for physics calc
  }
}
```

4.2 Drop Logic

- **Canvas:** Creates a new `SPINE` or `SATELLITE` node with the `media_in` / `media_out` values frozen from the payload.
- **Bucket:** Creates a new un-anchored node in the Bucket sidebar.

---

5. Live Node Editing (Canvas Focus)

When an existing Node is selected, the Inspector enables "Ripple Editing" (See Doc E).

5.1 Interaction Rules

- **Visual Cue:** The Mini-Timeline Range Bar turns **Green**.

- **Data Binding:** Changes write directly to the SQLite `nodes` table (debounced 500ms).

Action	Logic	Effect
Trim Head	Move In-Point Handle	`media_in_point` updates. Duration shrinks/grows.
Trim Tail	Move Out-Point Handle	`media_out_point` updates. Node B slides (Ripple).
Slip Media	Drag Range Bar Center	`in` and `out` shift equally. Duration locked. Node structure unchanged.

Doc J: Multicam & Sub-Node Isolation

Doc J: Multicam & Sub-Node Isolation Protocol

Version: 4.0

1. Executive Summary

The Concept: A Multicam Node is a Container. To the main Graph, it looks like a single generic block with a fixed duration. Inside the container exists a "Shadow Graph" (The Angles).

The Interaction: Users can edit in "Live Mode" (switching angles while playing) or "Isolation Mode" (opening the container to sync/adjust).

The Critical Action: "Flattening" is the act of destroying the Container and replacing it with a sequential chain of standard Clips based on the switching decisions.

---

2. Data Structure (The State Map)

The Multicam Node does not store the edit as physical nodes (until flattened). It stores them as a lightweight JSON map in `nodes.internal_state_map`.

Schema Definition:

```
JSON

{
  "active_audio_master": "A", // Which Angle drives the audio waveform
  "sync_method": "TIMECODE",  // How angles were aligned
  "switches": [
    { "time_offset": 0, "angle_id": "A" },
    { "time_offset": 4500, "angle_id": "B" }, // Switch to Cam B at 4.5s
    { "time_offset": 9200, "angle_id": "A" }
  ],
  "angles": {
    "A": "asset_uuid_1",
    "B": "asset_uuid_2",
```

```
    "C": "asset_uuid_3"
  }
}
```

---

3. The Two Interaction Modes

3.1 Live Mode (The Switcher)

- **Context:** The user is on the Main Canvas.
- **Behavior:** The Multicam Node plays back. Clicking 1/2/3/4 on the keyboard updates the `switches` array in real-time.
- **Graph Physics:** The Node's width (Duration) remains constant. No topological changes occur. Satellites attached to this node remain anchored to the Container.

3.2 Isolation Mode (The Sub-Graph)

- **Context:** User double-clicks the Multicam Node.
- **UI Behavior:** The Main Canvas "Dives Down." A breadcrumb appears: `Main Story > Interview Multi`.
- **The View:** The user sees a "Sync Map"—a stack of tracks showing all angles aligned by Timecode/Waveform.
- **Allowed Actions:**
    - Slip/Slide individual angles to fix sync.
    - Color Grade specific angles.
    - **Prohibited Actions:** You cannot add "Story Nodes" (images/text) inside a Multicam Container. It is strictly for Sync and Switching.

---

4. The Flattening Protocol (The Topological Explosion)

Flattening is a destructive operation that replaces **Node X** (1 Object) with **Nodes X1, X2, X3...** (N Objects).

The Rules of Engagement:

Rule A: The Anchor Hand-off

When the Container dissolves, the *first* resulting clip (X1) inherits the Container's `anchor_id` and `anchor_type`.

- *Before:* Container $\rightarrow$ Anchored to Parent A.

- *After:* Clip X1 $\rightarrow$ Anchored to Parent A. Clip X2 $\rightarrow$ Anchored to X1 (Append). Clip X3 $\rightarrow$ Anchored to X2.

Rule B: The Satellite Transfer (The "Gravity" Logic)

If the Multicam Container had a Note/Satellite attached at `00:00:15`:

1. The System calculates which *Switch Segment* encompasses `00:00:15`. (e.g., It falls inside Clip X3).
2. The Satellite is detached from the Container.
3. The Satellite is re-attached to **Clip X3**.
4. **Crucial Math:** The Satellite's `drift` (offset) is recalculated relative to Clip X3's start, not the Container's start.

Rule C: The Audio Paradox

- **If "Audio Follows Video":** The flattened clips refer to their specific angle's audio.
- **If "Audio Split":** The system generates *two* rows.
    1. **Row 1 (Spine):** The Video Cuts (Muted).
    2. **Row 2 (Parallel/Secondary):** A continuous Audio Node (The Master Audio Track) running underneath.

---

5. The Algorithm: `EXECUTE_FLATTEN(NodeID)`

When the user selects "Flatten Multicam":

1. **Transaction Start:** Open a new entry in `transaction_log` (Action: `FLATTEN_MULTICAM`).
2. **Read Map:** Parse `internal_state_map`.
3. **Generate Nodes:**
    - Iterate through the `switches` array.
    - For each switch, create a new `nodes` entry (Type: `SPINE`).
    - Calculate `media_in` and `media_out` based on the offsets.
4. **Link Topology:**
    - Set `parent_id` for the first new node to the Container's `parent_id`.
    - Daisy-chain the subsequent nodes (`parent_id` = previous node).
5. **Migrate Children:**
    - Find all nodes where `parent_id` == `ContainerID`.
    - For each Child, calculate its absolute time position.
    - Find which new "Switch Node" occupies that time.
    - Update Child's `parent_id` to the new Node ID.
    - Update Child's `drift`.
6. **Cleanup:** Delete the original Container Node.
7. **Commit:** Close Transaction.

---

6. Edge Case: The "Un-Flatten"

Since we destroyed the Container, how do we Undo?

- **Logic:** We rely entirely on the `transaction_log`.
- **Process:** The `undo_payload` contains the full JSON snapshot of the original Container Node. The system deletes the chain of clips (X1...Xn) and restores the single Container Node object.

Doc K: Project Lifecycle

Doc K: Project Lifecycle

Version: 1.0 (Dashboard & Canvas Logic)

---

1. The Launch Experience (Dashboard)

Upon launching Story Graph, the user lands on the **Project Dashboard**. This is a persistent window that reads from the local `projects` table.

1.1 The Card Grid

- **Query:** `SELECT * FROM projects ORDER BY last_edited_at DESC`.
- **Visual:** A grid of cards.
- **Card Metadata:**
    - **Image:** `image_path` (or default placeholder if NULL).
    - **Title:** `name`.
    - **Client:** `client_name` (Subtitle).
    - **Date:** "Edited 2h ago" (Relative Time).
    - **Status:** Badge (e.g., "In Progress" = Green, "Archived" = Grey).

1.2 Context Menu (Right-Click)

Option	Action	Logic
Open	`ipc.invoke('project-load', { id })`	Routes to Editor View.
Edit Details	Opens Modal.	Updates `name`, `client`, etc.
Duplicate	Deep Copy.	Clones Project Row + All linked tables (Nodes, Media).
Archive	Update Status.	Hides from main view unless filter is "All".

Delete	**Destructive.**	`DELETE FROM projects...` (Cascade deletes all data).

---

2. Project Creation Modal

Trigger: Big "New Project" button on Dashboard.

2.1 Input Fields (Required & Optional)

Field	UI Input	DB Column	Notes
Name	Text Input	`name`	**Required.**
Client	Text Input	`client_name`	
Company	Text Input	`company_name`	
Cover Img	File Drop	`image_path`	Copied to `/App_Data/Project_Images/UUID.jpg`.
FPS	Dropdown	`default_fps`	"House Standard" (e.g., 23.976). Critical base.
Resolution	Dropdown	`default_resolution`	e.g., "1920x1080".

2.2 The Initialization Sequence

1. **Generate UUID:** Create unique Project ID.
2. **Insert Row:** Write data to `projects` table.
3. **Create Root Bin:** Insert row into `bins` (`name='Master'`, `parent_id=NULL`).
4. **Create Default Canvas:** Insert row into `canvases` (`name='Main Canvas'`).
5. **Route:** Close Modal $\to$ Open Editor View (Canvas).

---

3. The Editor Workspace (Canvas Management)

A Project can contain multiple "Canvases" (Timelines/Storyboards).

3.1 The Canvas Tab Bar

- **Location:** Top of the Center Panel.
- **Function:** Switch between different active graphs.
- **Action:** Clicking a tab triggers `ipc.invoke('graph-get-state', { canvasId })` to re-render the topology.

3.2 Canvas Settings Inheritance

When creating a *new* Canvas inside a Project:

- **Checkbox:** "Use Project Settings" (Default: Checked).
- **Logic:**
    - **True:** `canvas.fps` = `project.default_fps`.
    - **False:** User manually selects specific FPS (e.g., for a 60fps Social Media cut).
- **Why:** This ensures the "Temporal Floor" grid aligns with the master project settings by default, preventing drift errors.

---

4. Data Persistence Strategy (Auto-Save)

Story Graph is **Local-First**, meaning there is no "Save Button" anxiety.

4.1 The Write Trigger

- **Event:** Any transactional change (Node Move, Trim, Text Edit).
- **Action:** Immediate synchronous write to SQLite (WAL Mode).
- **Debounce:** Text inputs (Description, Transcript) are debounced (500ms) to prevent DB thrashing.

4.2 The "Last Edited" Update

- Logic: Every write operation triggers a side-effect update:
  UPDATE projects SET last_edited_at = CURRENT_TIMESTAMP WHERE id = ?.
- **Result:** The Dashboard always sorts the most active project to the top.

---

5. Application Lifecycle

- **On Quit (`Cmd+Q`):**
    1. Main Process catches `app.on('before-quit')`.

2. Closes all active DB connections.
3. Writes a "Clean Exit" log entry.
4. Terminates process.

Doc L: Export Protocol (FCPXML)

Doc L: Export Protocol (FCPXML)

Version: 1.5 (FCPXML v1.10 Standard)

---

1. Export Philosophy

Story Graph is a "Rough Cut" architecture tool. It is not a finishing tool. Therefore, the export process is not just saving a file; it is a **Translation Layer**.

- **Input:** A non-linear, hierarchical Graph (`nodes` table).
- **Process:** Flattening the topology into linear tracks.
- **Output:** A standardized `FCPXML` (v1.10) file compatible with Resolve and FCP.

---

2. The Translation Algorithm

The export engine must convert our internal "Drift" logic into absolute "Rational Time".

2.1 Step 1: Flattening the Graph

We use the same logic as the **Temporal Floor** (Doc E) to generate a virtual linear timeline in memory.

- **Input:** Current Canvas ID.
- **Process:**
    1. Calculate `Timeline_In` / `Timeline_Out` for every node.
    2. Determine `Track_Index` (V1, V2, V3) using the Auto-Layout logic (Doc F).
- **Result:** A list of `ClipObjects`:
- TypeScript

```
{
  assetPath: "/Users/Me/Video.mov",
  start: 3600,   // Timeline start (frames)
  duration: 120, // Length (frames)
  offset: 0,     // Source In-Point (frames)
  lane: 1        // Track V1
}
```

-
-

2.2 Step 2: Rational Time Conversion

XMLs do not use floats or milliseconds. They use rational fractions (Numerator/Denominator).

- **Formula:** `FrameCount / Timebase`.
- **Example:**
    - Project FPS: 23.976 (Timebase: 24000/1001).
    - `start="3600/24000s"` (Incorrect).
    - **Correct FCPXML Format:** `start="150150/1000s"` (Rational string).

---

3. The FCPXML Structure

3.1 Header (Resources)

We must declare every unique asset used in the cut.

```
XML

<resources>
  <format id="r1" name="FFVideoFormat1080p2398" frameDuration="1001/24000s" width="1920" height="1080"/>
  <asset id="r2" name="CamA_01" src="file:///Users/Me/CamA_01.mov" start="0s" duration="100s" hasVideo="1" format="r1"/>
</resources>
```

3.2 Body (Sequence)

The structure uses a "Spine" approach similar to our Graph.

- **Spine:** Corresponds to our **Track V1**.
- **Connected Clips:** Corresponds to our **Satellites (V2, V3)**.

```
XML

<spine>
  <video ref="r2" offset="0s" duration="5s" start="10s">
     <video ref="r3" lane="1" offset="2s" duration="3s" start="0s"/>
  </video>
</spine>
```

- **Translation Note:** While Story Graph allows "Drift", FCPXML requires connected clips to be anchored to a specific point in the parent.
    - `offset`: The position relative to the parent's start.

---

4. Data Mapping Table

Story Graph Data	FCPXML Attribute	Conversion Logic
projects.default_fps	frameDuration	24 $\to$ "100/2400s", 23.976 $\to$ "1001/24000s".
media_library.file_path	src	Prepend file:// protocol.
nodes.media_in_point	start	Convert Seconds $\to$ Rational String.
nodes.duration	duration	Convert Seconds $\to$ Rational String.
nodes.ui_track_lane	lane	V1 $\to$ Spine (No Lane), V2 $\to$ lane="1", V3 $\to$ lane="2".

---

5. Handling Complexities

5.1 Gaps (Spacers)

- **Problem:** Our Graph allows empty space between Node A and Node B (Drift). XML Spines must be continuous.
- **Solution:** Insert <gap> elements.
    - If Node A ends at 00:05 and Node B starts at 00:07, inject a 2-second <gap> element between them in the XML Spine.

5.2 Multicam Flattening

- **Rule:** XML export **ALWAYS** flattens Multicam nodes.

- **Process:** The system reads the `internal_state_map` (Doc I) and generates individual `<video>` clips for each cut, placing them sequentially on the Spine. It does *not* export a "Multicam Clip" container to the NLE.

---

6. Implementation (Main Process)

- **Trigger:** `ipcMain.handle('project-export-xml', { canvasId, filePath })`
- **Library:** Use `xmlbuilder2` (Node.js) to construct the tree programmatically.
- **Validation:** Validation against the FCPXML DTD (Document Type Definition) to ensure the file will actually open in Resolve.