SVG Embedded Semantic Metadata (SESM)

A standard for embedding structured metadata and instructions directly inside SVG files — turning visual assets into self-describing, machine-readable artifacts.

DRAFT V0.2 | ASSET-LEVEL METADATA | APTLANTIS STUDIO

What Is SESM?

The Core Idea

SESM defines a simple, implementation-agnostic way to embed structured metadata inside SVG assets. SVGs become more than images — they carry semantic context wherever they go.

SVGs as Multi-Role Artifacts

- Visual assets and metadata carriers
- Semantic capsules and archive-friendly documentation
- LLM-readable context packets
- UI runtime hints
- Generated artifacts with provenance

Why SESM Matters for Aptlantis Studio

In Aptlantis Studio, many visible interface components are **compiled SVG artifacts** generated from JSON or JSONL manifests. A single SVG can represent a dataset card, pipeline status panel, theme board, stats panel, navigation artifact, dataset identity badge, or a crawler-facing semantic summary.

Dataset Card

Structured identity for curated datasets

Pipeline Panel

Status and provenance for build pipelines

Theme Board

Visual system palette and semantic roles

Crawler Summary

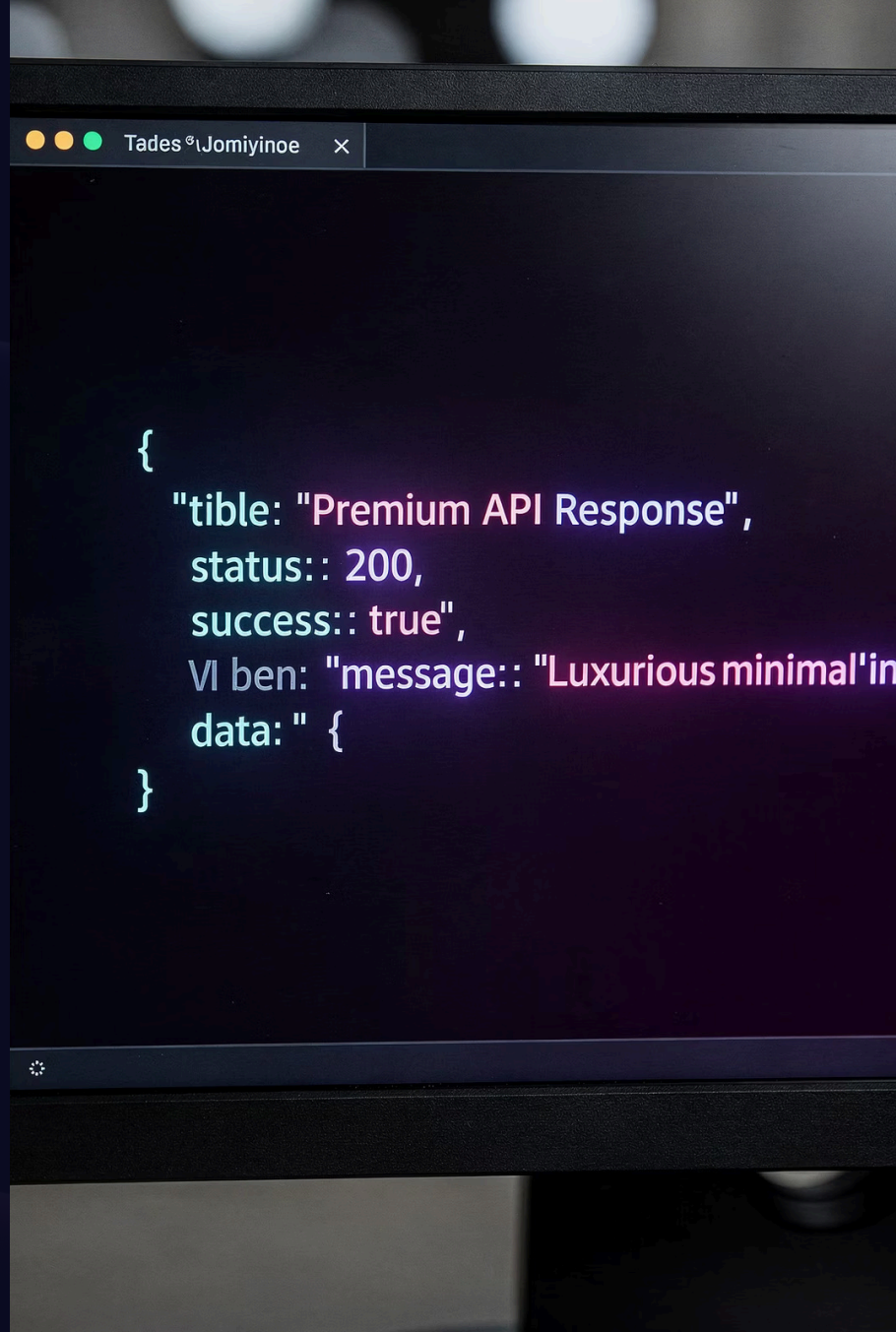
Indexing hints embedded in the asset itself

Design Philosophy

> Put a valid JSON object inside an SVG <metadata> element. Everything else is convention.

SESM is intentionally simple. This simplicity is what allows it to survive static hosting, mirroring, scraping, archival, direct filesystem copying, Internet Archive uploads, local agent indexing, page extraction, and build pipeline regeneration.

SESM should be easy to generate from a small Rust, Go, Python, or JavaScript utility — no heavy framework required.



SESM Goals

1. **Embed Metadata**
   Structured metadata lives directly in the SVG asset
2. **Preserve Context**
   Semantic meaning survives separation from the original page
3. **Crawler & Archival Hints**
   No need to fetch large external documents
4. **LLM Summaries**
   Usage hints for language models and local agents
5. **UI Runtime Hints**
   Cards, panels, datasets, themes, and generated components
6. **Provenance & Rebuild**
   Supports deterministic artifact regeneration

Non-Goals

SESM provides **semantic hints, not mandates**. It is explicitly not intended to:

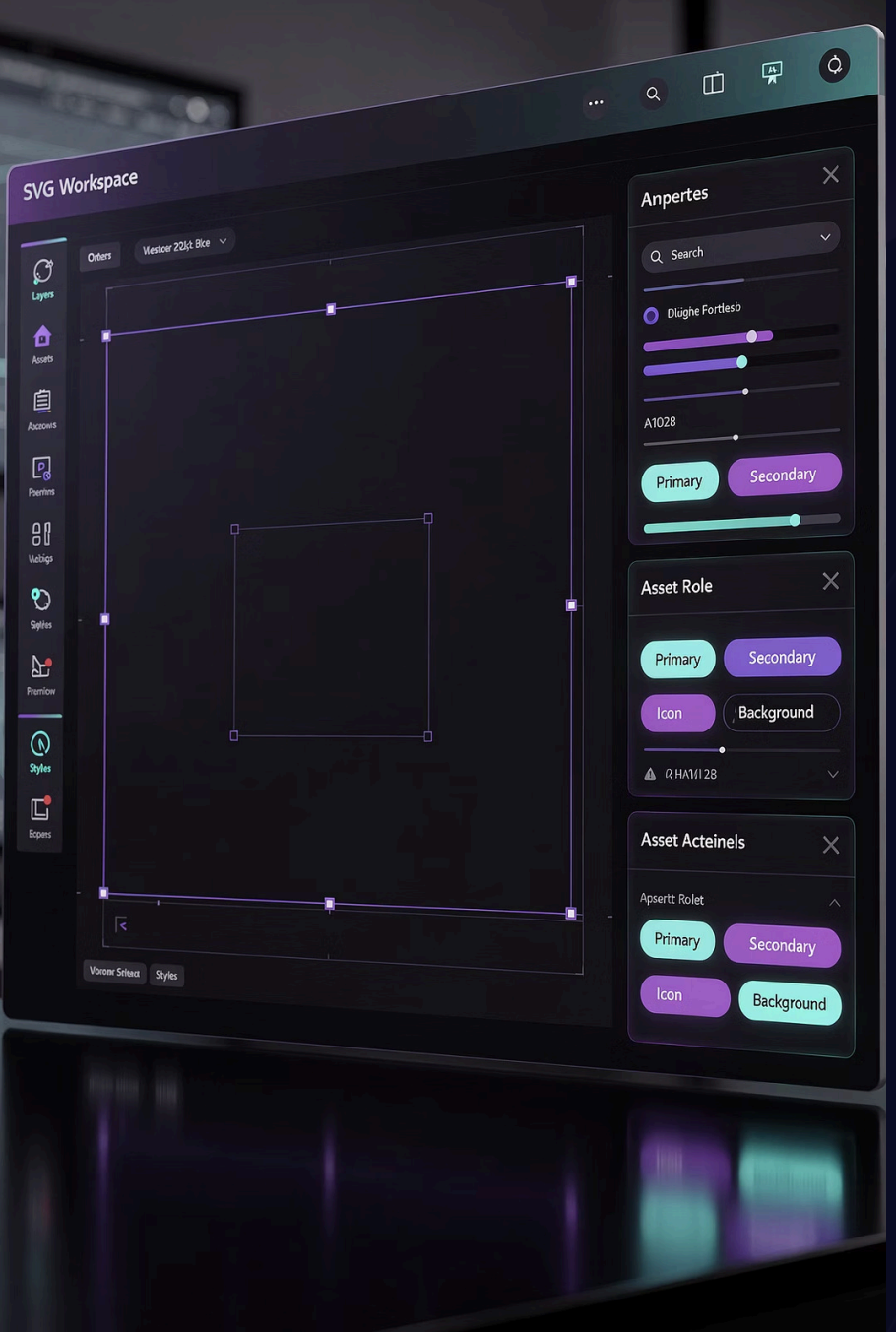
- Replace JSON-LD in HTML or site maps / robots.txt
- Contain secrets, credentials, or internal-only data
- Define a complete ontology for all visual assets
- Require any specific AI, LLM, crawler, or frontend framework
- Enforce rendering behavior or act as access control
- Require JavaScript to be useful

SESM Block Structure

A SESM block is a single JSON object. Only `sesm_version` is required. Agents must ignore unknown fields rather than failing.

```
{
  "sesm_version": "0.2.0",
  "asset":      {},
  "artifact":   {},
  "theme":      {},
  "ui":         {},
  "crawl":      {},
  "llm":        {},
  "links":      {},
  "provenance": {},
  "integrity":  {},
  "extra":      {}
}
```

Future revisions follow semantic versioning: MAJOR for breaking changes, MINOR for additive optional fields, PATCH for documentation clarifications.



The asset & artifact Objects

asset — What the SVG Represents

Describes the SVG's identity: `id`, `role`, `title`, `description`, `ecosystem`, and `tags`. Common roles include `dataset-card`, `theme-board`, `pipeline-panel`, `stats-panel`, `logo`, `icon`, and `decorative`.

artifact — Provenance of the Generated File

Describes the SVG as a compiled artifact: `kind`, `artifact_id`, `source_type`, `source_id`, `template_id`, `output_path`, and `build_profile` (dev / preview / production).

The theme Object & NIPC Semantic Colors

The `theme` object describes the visual theme and semantic color mapping. For Aptlantis Studio, palette semantics come from the **NeonInkPaletteContract (NIPC)**.

Clarity / Orientation

`info`, `structure`, `navigation` — Understanding and wayfinding

Trust / Validation

`success`, `verified`, `stable` — Evidence-backed confidence

Risk / Constraint

`critical`, `error`, `blocked` — Failure, blockers, hard limits

Creation / Build / Code

`code-heat`, `build`, `tooling` — Rust, execution, generated output

The ui Object — Runtime Placement Hints

The `ui` object provides hints for runtime placement and layout. These are hints, not requirements.

Key Fields

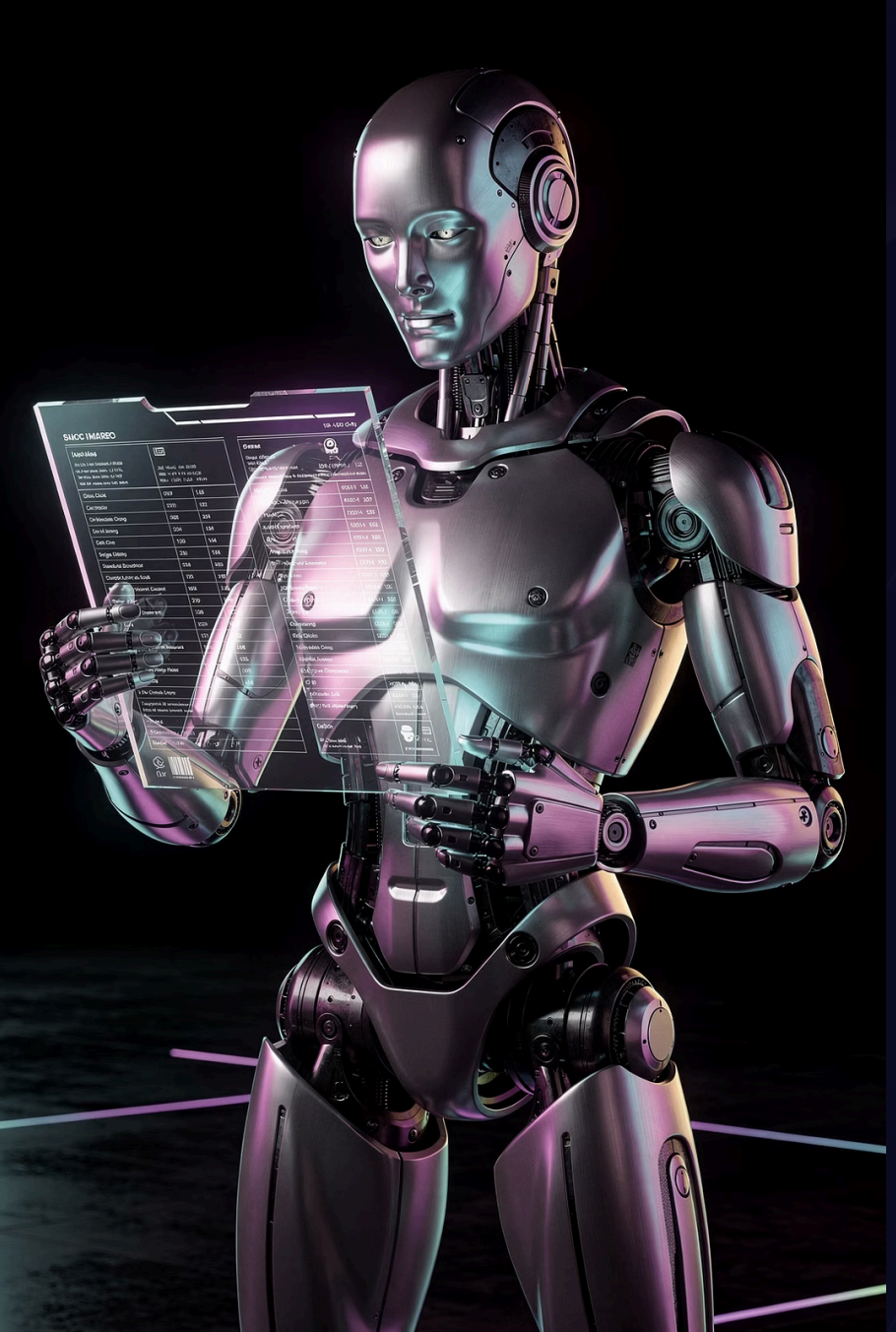
- `component_type`: panel, card, badge, hero, graph-node
- `preferred_layout`: artifact-card, logo-left-text-right, centered
- `preferred_regions` / `avoid_regions`
- `responsive_behavior`: preserve-aspect-ratio, crop-safe, scale-to-fit

Interaction Hints

- `click_target`: canonical_html
- `hover_behavior`: show-metadata
- `supports_focus`: true/false

Dimensions

Intended `width`, `height`, and `aspect_ratio` for the rendered component.



crawl & llm Objects

crawl — Indexing & Archival Hints

`indexable` and `archive` booleans signal intent. `discover_paths` lists related paths worth crawling. `canonical_group` provides logical grouping (e.g., datasets, themes, pipelines). SESM does not replace robots.txt — crawlers must still respect host policy.

llm — Language Model Context

`summary` provides a short description. `intended_interpretation` clarifies what the asset represents. `prompt_hints` guide LLM context-building. `avoid_interpretations` prevents common agent mistakes. Prefer explicit SESM fields over visual color inference.

links, provenance & integrity

links

Connects the SVG to related resources: `canonical_html`, `jsonld`, `manifest`, `download`, `torrent`, `source_repo`, `license`, `snapshot_hashes`.

provenance

Records how the SVG was generated: generator name/version/language, `generated_at` (ISO-8601), `input_records`, `reproducible` flag, optional `git_commit` and `build_id`.

integrity

Non-secret integrity metadata: `content_hash` (e.g., blake3-256), path to `manifest` and `signed_manifest`. Not a replacement for signed manifests — prefer linking over embedding large signatures.

The extra Object & Vendor Extensions

The `extra` object is a namespace for platform-specific extensions. Vendor extensions must be namespaced to avoid collisions.

```
{
  "extra": {
    "vendor": {
      "aptlantis": {
        "schema_id":      "DatasetCard",
        "schema_version": "1.0.0",
        "platform":       "aptlantis-studio",
        "domain":         "www.aptlantis.studio"
      }
    }
  }
}
```

Recommended pattern: always nest under `extra.vendor.{organization_or_platform_name}` to prevent field collisions with future SESM versions.

Embedding SESM in SVG

SESM metadata must be embedded inside an SVG `<metadata>` element with `id="sesm"`. Only one SESM block per SVG. Agents parse the first and ignore the rest.

CDATA Form (Recommended)

Use when templates are processed by XML-aware tools or special characters may appear in JSON strings.

```
<metadata id="sesm"><![CDATA[
{
  "sesm_version": "0.2.0",
  "asset": { "role": "dataset-card" }
}
]]></metadata>
```

Raw JSON Form

Acceptable when the toolchain safely preserves the JSON without XML escaping issues.

```
<metadata id="sesm">
{
 "sesm_version": "0.2.0"
}
</metadata>
```

Aptlantis Studio: Dataset Card Example

A real-world SESM block for the **Rust Code Corpus** dataset card, combining asset identity, artifact provenance, Neon Ink theme, UI hints, crawl permissions, and LLM context in a single embedded JSON object.

1

asset

id: `rust-code-corpus-card`, role: `dataset-card`, tags: rust, fine-tuning, small-models

2

artifact

kind: `compiled-svg`, source: `datasets.jsonl`, template: `dataset-card-v1`, build: production

3

theme

Neon Ink v0.1, dark mode, accent: `code-heat #F97316`, state: active, glow: soft

4

llm

Summary: curated Rust fine-tuning dataset. Hint: prefer SESM state fields over visual color inference.

Aptlantis Studio: Theme Board Example

The **Neon Ink Theme Board** SVG uses SESM to document the full palette and semantic family structure of the Aptlantis Studio visual system.

Token Palette

void #050816, base #0B0F1A, panel #111827, info #22D3EE, process #A78BFA, featured #F472B6, success #34D399, important #FACC15, critical #F43F5E, code-heat #F97316

Semantic Families

Nine families: clarity-orientation, trust-validation, attention-learning-anchor, risk-constraint, process-transformation, creation-build-code, discovery-creative-featured, research-experimental, canonical-archive-neutral.

Validation Rules & Agent Behavior

Recommended Validation Checks

1. SVG contains `<metadata id="sesm">`
2. Contents parse as valid JSON
3. `sesm_version` exists and is a string
4. Known top-level fields are objects
5. URLs are strings; hex colors follow `#RRGGBB`
6. `crawl.indexable` and `provenance.generated` are booleans
7. No obvious secret-like fields present

> ⓘ Validators should warn, not fail, for unknown fields.

Agent Behavior

1. Locate and extract `<metadata id="sesm">`
2. Parse contents as JSON; check `sesm_version`
3. Use known fields; ignore unknown fields
4. Prefer explicit SESM fields over visual inference
5. Use `links.canonical_html` for full page context
6. Use `llm.summary` and `prompt_hints` to avoid guessing

Security & Privacy Considerations

SESM is descriptive, not protective. Assume SESM blocks may be mirrored, scraped, indexed, archived, exposed to LLMs, or copied outside their original context.

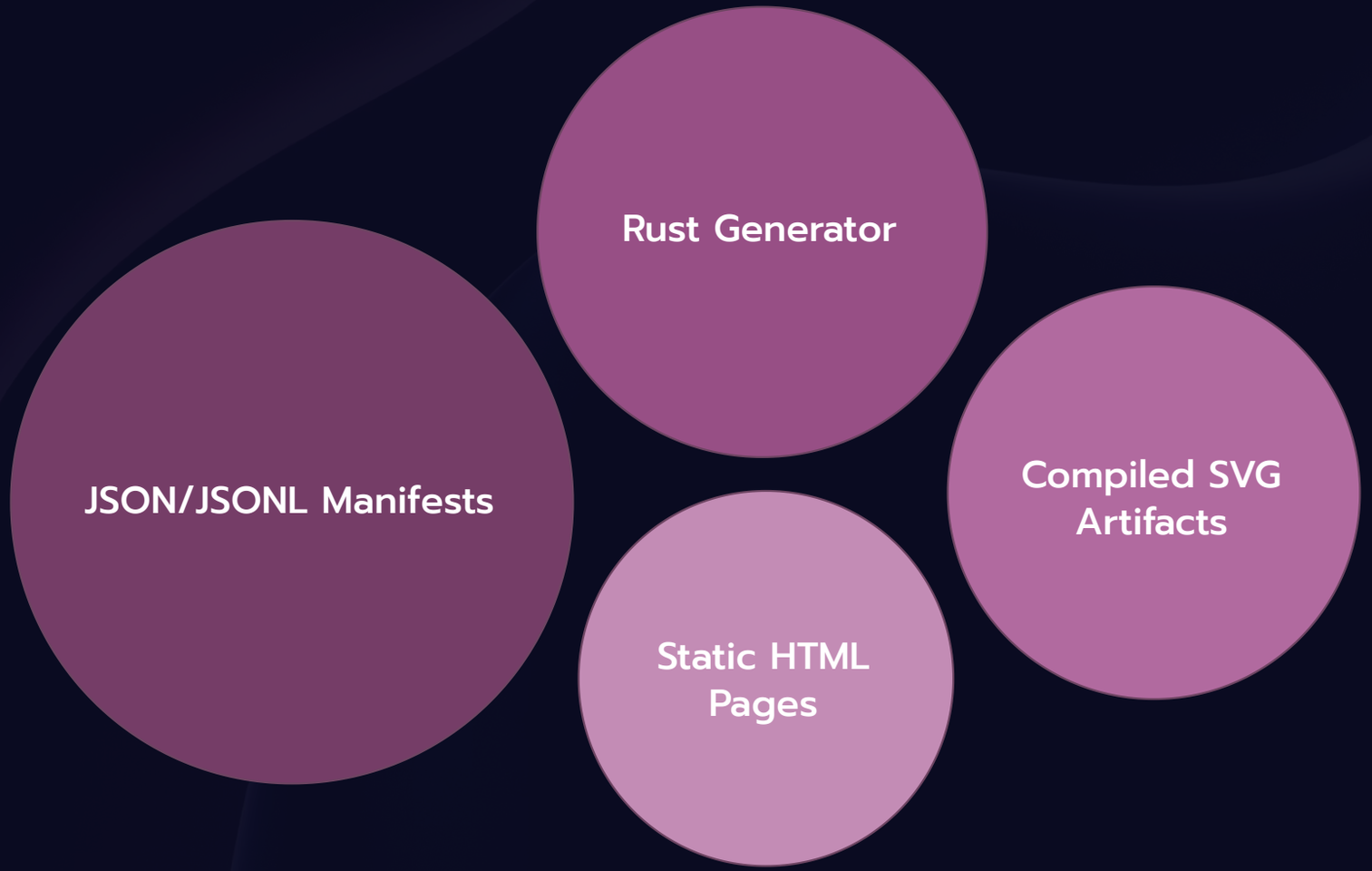
Never Embed

API keys, tokens, private URLs, credentials, unreleased project names, private filesystem paths, personal information, or internal-only notes.

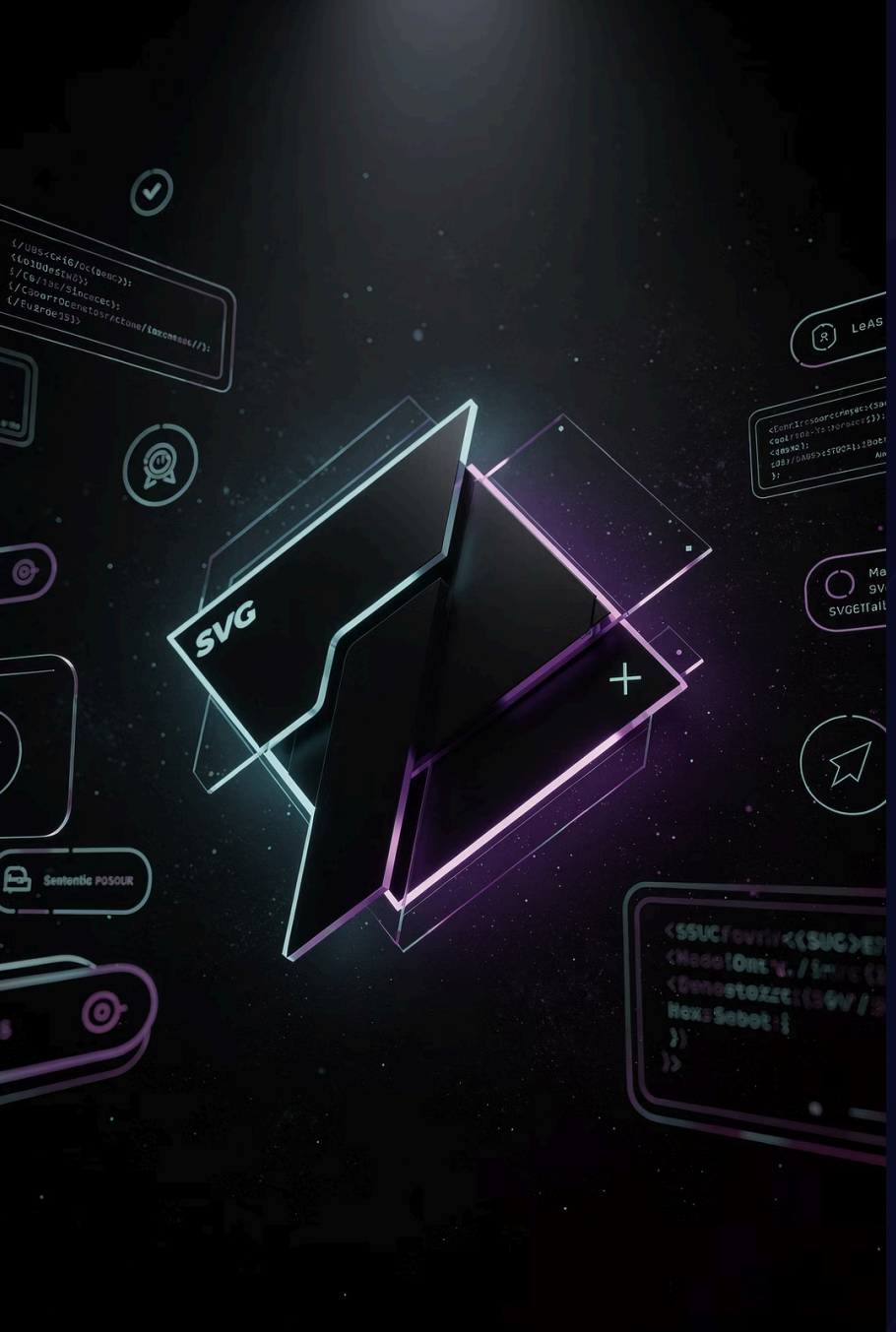
Keep Private Behavior In

Backend configs, signed manifests, and access control systems — not in SESM blocks that travel with the SVG file.

Build Pipelines, Rust Generators & Compatibility



The generator stamps each SVG with source record ID, template ID, generator name/version, timestamp, canonical output path, theme ID, semantic state, and crawl/LLM hints. SESM v0.2 remains backward-compatible with v0.1-style blocks — agents must not require every v0.2 field to be present.



Summary: SVGs Are Self-Describing Artifacts

SESM is an intentionally simple convention: **a valid JSON object embedded inside**