ENGINEERING STANDARDS LIBRARY / SOFTWARE ENGINEERING

Semantic UI Protocols and Typed Renderer Abstraction Standard

Semantic intent payloads, typed renderers, deterministic presentation, accessibility, and model-independent interface contracts.

PERMANENT DOCUMENT ID

MYTHOS-SWE-0008

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

Semantic UI Protocols and Typed Renderer Abstraction Standard

DOCUMENT ID
MYTHOS-SWE-0008

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

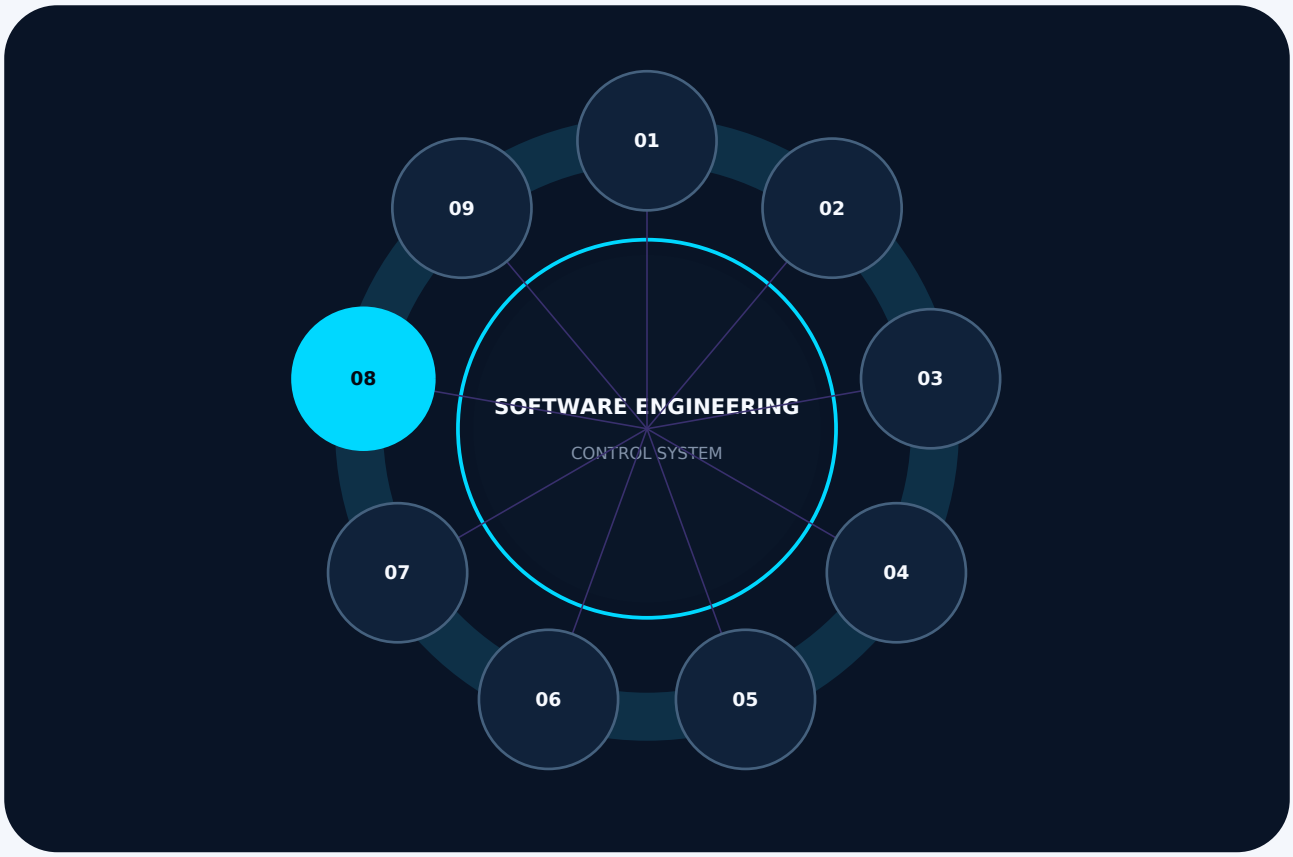
SCHEDULED REVIEW
2027-01-24

16	6	4	1
ATOMIC CRITERIA	ASSURANCE VIEWS	IMPLEMENTATION PROFILES	PERMANENT IDENTITY

Publication doctrine. This publication establishes durable guidance or requirements for its defined scope. Interpretation must preserve the permanent document identity, recorded evidence, controlled exceptions, and formal revision history.

DOMAIN CONTROL SYSTEM

MYTHOS-SWE-0008 inside the software engineering standard constellation



Connected assurance	Architecture, implementation, operations, evidence, and lifecycle are evaluated as one controlled system.
Specialization rule	Companion standards may add precision, but they cannot silently weaken this publication's requirements.

EXECUTIVE POSITION

Semantic UI Protocols and Typed Renderer Abstraction Standard

The integration of Large Language Models with user interfaces has failed.

Evidence-bearing delivery path

Assurance maturity spectrum

CONTENTS

Publication navigation

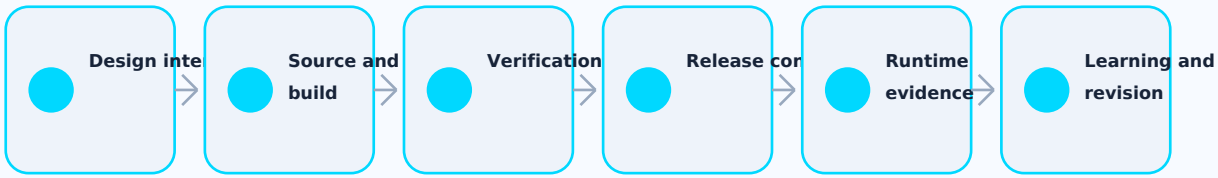
CONTROL TOPOLOGY

Assurance model and evaluation surface

The standard is assessed as an evidence-bearing system. Capability scores do not replace mandatory gates, implementation proof, or operational validation.

Capability family	Evaluation nodes
Semantic Payload Contracts (Weight: 25%)	2
Renderer Abstraction and Multi-Target (Weight: 20%)	2
Security and Injection Prevention (Weight: 20%)	2
High-Density and Accelerated Rendering (Weight: 15%)	2
State and Lifecycle Management (Weight: 10%)	1
Operational Lifecycle (Weight: 10%)	1
Suite Composition	2
Additional Evaluation Dimension	4

Lifecycle control loop



EVALUATION INDEX

Criteria and evidence map

#	Capability family	Criterion
1	Semantic Payload Contracts (Weight: 25%)	Intent vs. Markup
2	Semantic Payload Contracts (Weight: 25%)	Payload Schema Governance
3	Renderer Abstraction and Multi-Target (Weight: 20%)	Renderer Trait
4	Renderer Abstraction and Multi-Target (Weight: 20%)	Cross-Medium Fidelity
5	Security and Injection Prevention (Weight: 20%)	Markup Injection Prevention
6	Security and Injection Prevention (Weight: 20%)	Payload Validation
7	High-Density and Accelerated Rendering (Weight: 15%)	Accelerated Graphics
8	High-Density and Accelerated Rendering (Weight: 15%)	Data Density
9	State and Lifecycle Management (Weight: 10%)	UI State Determinism
10	Operational Lifecycle (Weight: 10%)	UI Telemetry
11	Suite Composition	M-SUI-S-Security
12	Suite Composition	M-SUI-S-Rendering
13	Additional Evaluation Dimension	UI Architecture Dimensions
14	Additional Evaluation Dimension	The UI Doctrine
15	Additional Evaluation Dimension	Scoring Model
16	Additional Evaluation Dimension	Suite Composition

NORMATIVE PUBLICATION

Requirements, evaluation methodology, and implementation guidance

The following body preserves the substantive source standard while normalizing publication status, identity, navigation, and visual presentation.

Executive Mandate

The integration of Large Language Models with user interfaces has failed.

Current architectures treat LLMs as HTML generators, piping raw markdown or hallucinated JSX directly into the DOM via `dangerouslySetInnerHTML`. This conflation of reasoning (the model) and rendering (the UI) creates catastrophic security vulnerabilities (XSS via prompt injection), framework lock-in (React-specific components), and cognitive overload (walls of text instead of structured data).

This standard exists because:

1. The LLM is Untrusted Markup: An LLM that outputs HTML or markdown is a Cross-Site Scripting (XSS) vector. The model MUST output structured, typed *intent* (e.g., "Render an approval dialog with these options"), not *markup* (e.g., `...`).
2. Markdown is a Liability for Complex Data: Rendering network topologies, firewall blast radii, or agent state machines as markdown tables is an affront to human cognition. High-density data requires accelerated rendering (WebGPU/wgpu).
3. Framework Coupling is Architectural Suicide: Hardcoding the UI to React, Svelte, or Swift ties the system's lifespan to a framework's hype cycle. The renderer MUST be an abstract trait; the framework is a pluggable detail.
4. The Medium is Variable: An agent's output must be renderable on a browser, a Tauri desktop app, a CLI terminal, or an XR headset without rewriting the agent's output logic.

This document establishes the Mythos Semantic UI Standard (MSUIS), a comprehensive, evidence-based methodology for evaluating the protocols between AI and UI, the abstraction of renderers, and the fidelity of high-density data visualization.

Scope and Applicability

In Scope

This standard covers the evaluation of:

1. Semantic UI Protocols (LLM-to-Renderer typed payloads)
1. Typed Renderer Abstractions (multi-target: Web, Desktop, CLI, XR)
1. High-density data visualization and accelerated rendering (WebGPU, wgpu)
1. Spatial computing UI integration (WebXR, 3D engines)

1. UI state machines and deterministic rendering
1. Security boundaries between LLM output and the DOM

Out of Scope

1. General desktop architecture (covered in MYTHOS-SWE-0006)
1. AI Avatar and Persona design (covered in MYTHOS-AI-0010)
1. General accessibility (covered in MYTHOS-UX-0001)

Audience

1. Frontend engineers and UI architects
1. AI/ML engineers integrating LLMs with user interfaces
1. Security teams evaluating XSS and injection surfaces
1. Platform engineers building multi-target UI systems
1. Standards bodies seeking a reference semantic UI methodology

Definitions and Taxonomy

UI Architecture Dimensions

Dimension	Definition
Semantic Payload	A structured, typed data object (e.g., JSON/Protobuf) emitted by the LLM describing *what* should be rendered, not *how*.
Typed Renderer	An abstract interface that consumes Semantic Payloads and renders them appropriately for the target medium (Browser, Desktop, CLI, XR).
High-Density Visualization	The rendering of complex, interconnected data (e.g., network graphs, state machines) using accelerated graphics (WebGPU) rather than DOM elements.
Cyberpunk-Noir Aesthetic	A visual design philosophy prioritizing maximum data density, high contrast, minimal chrome, and accelerated rendering, optimized for operator cognition in critical infrastructure.

The UI Doctrine

> The LLM emits intent, not markup. The renderer is a trait, not a framework. Markdown is for prose; WebGPU is for infrastructure. A hallucinated tag is a security breach.

Evaluation Methodology

Scoring Model

Each capability is scored from 0 to 5.

Score	Meaning
0	Fails basic task; raw markdown/HTML, framework-coupled, DOM-heavy

Score	Meaning
1	Basic component libraries but untyped LLM output
2	Functional but lacks semantic protocols or renderer abstraction
3	Solid production performance; semantic payloads, typed renderers, basic acceleration
4	Strong performance; multi-target, WebGPU, spatial awareness
5	Best-in-class; sets the standard for deterministic, high-density semantic rendering

Weighting

Category	Weight	Rationale
Semantic Payload Contracts	25%	Unstructured LLM output is an XSS vector and a maintenance nightmare
Renderer Abstraction & Multi-Target	20%	Framework lock-in kills agility
Security & Injection Prevention	20%	The DOM is untrusted; LLM output is untrusted
High-Density & Accelerated Rendering	15%	DOM cannot render infrastructure data at scale
State & Lifecycle Management	10%	UI state must be deterministic and replayable
Operational Lifecycle	10%	UI must be governed continuously

Total: 100%

A system MUST score at least 3.0 in Semantic Payloads and Security to be considered for production use.

Evaluation Categories

Semantic Payload Contracts (Weight: 25%)

Intent vs. Markup

Does the LLM emit semantic intent (typed payloads) rather than markup (HTML/JSX)?

Score	Criteria
0	LLM outputs raw HTML/JSX strings
1	LLM outputs markdown with embedded HTML
2	LLM outputs structured JSON but with styling/layout instructions
3	LLM outputs typed semantic payloads (e.g., `{ type: "ApprovalDialog", options: [...] }`)
4	Semantic payloads defined via schema (Protobuf/JSON Schema); validated before rendering
5	Perfect separation: formally verified semantic schemas, zero markup in LLM output, compile-time payload validation

Payload Schema Governance

Are semantic payload schemas versioned and backward compatible?

Score	Criteria
0	No schema; payload structure ad-hoc
1	Implicit schema defined by renderer implementation
2	Schema defined in shared types (TypeScript)
3	Schema defined in language-agnostic format (JSON Schema/Protobuf)
4	Schema versioning with automated compatibility checks in CI
5	Perfect governance: formally verified schema evolution, zero breaking renderer changes

Renderer Abstraction and Multi-Target (Weight: 20%)

Renderer Trait

Is the UI renderer an abstract trait/interface, decoupled from the LLM and business logic?

Score	Criteria
0	Business logic directly manipulates DOM/framework components
1	Renderer abstraction exists but leaks framework types
2	Clean renderer trait; framework (React/Svelte) is a pluggable implementation
3	Multiple renderer implementations (Web, Desktop/Tauri)
4	CLI/TUI renderer for headless/terminal environments
5	Perfect abstraction: formally verified renderer trait, zero business logic awareness of rendering medium

Cross-Medium Fidelity

Can the same semantic payload render effectively across different mediums?

Score	Criteria
0	Web-only; no other targets
1	Web and mobile web (responsive)
2	Web and Desktop (Tauri/Electron)
3	Web, Desktop, and CLI/TUI
4	Web, Desktop, CLI, and native notifications
5	Perfect fidelity: Web, Desktop, CLI, and XR (WebXR/Spatial) from the same semantic payload

Security and Injection Prevention (Weight: 20%)

Markup Injection Prevention

Is the rendering pipeline immune to LLM-driven XSS/injection?

Score	Criteria
0	`dangerouslySetInnerHTML` or equivalent used for LLM output
1	Output sanitized via DOMPurify or similar
2	Allowlist of safe HTML tags enforced
3	No raw HTML rendering; semantic payloads only
4	Sandboxed rendering context (iframe/WASM) for any untrusted content
5	Perfect security: formally verified zero-markup pipeline, cryptographic attestation of payload integrity, no dynamic code execution

Payload Validation

Are semantic payloads strictly validated before rendering?

Score	Criteria
0	No validation; renderer handles malformed data
1	Basic type checking (TypeScript)
2	Runtime validation (Zod/Joi) at the boundary
3	Schema-validated payloads with descriptive errors for LLM self-correction
4	Schema validation + feedback loop to LLM for retry on schema violation
5	Perfect validation: formally verified schemas, zero invalid renders, automated LLM self-correction loop

High-Density and Accelerated Rendering (Weight: 15%)

Accelerated Graphics

Is high-density data (graphs, topologies, state machines) rendered using GPU acceleration?

Score	Criteria
0	SVG/DOM-based rendering for complex data (laggy, limited scale)
1	Canvas 2D for specific visualizations
2	WebGL for primary data views
3	WebGPU (browser) / wgpu (desktop) for compute-shader-driven layout
4	WebGPU rendering with 1M+ data points at 60fps
5	Perfect acceleration: WebGPU/wgpu, formally verified render pipelines, cyberpunk-noir density, zero frame drops under load

Data Density

Can the UI display maximum information density without cognitive overload?

Score	Criteria
0	Low density; lots of whitespace, minimal data per screen
1	Standard dashboard layouts; moderate density
2	Dense tables and charts; high information per pixel
3	Cyberpunk-Noir aesthetic: dark mode, high contrast, dense telemetry, minimal chrome

Score	Criteria
4	Adaptive density based on user role and alert state
5	Perfect density: AI-driven semantic zoom, context-aware density scaling, formally verified cognitive load limits

State and Lifecycle Management (Weight: 10%)

UI State Determinism

Is the UI state a deterministic projection of the backend state?

Score	Criteria
0	UI state managed independently in frontend; drifts from backend
1	UI state synchronized via REST polling
2	UI state derived from streaming backend events (SSE/WebSocket)
3	UI state is a pure function of backend state (local-first event sourcing)
4	State transitions are typed and deterministic; time-travel debugging supported
5	Perfect determinism: formally verified UI state machine, zero state drift, cryptographic state hashing

Operational Lifecycle (Weight: 10%)

UI Telemetry

Is the rendering pipeline observable?

Score	Criteria
0	No UI telemetry
1	Error tracking (Sentry) only
2	Performance monitoring (Core Web Vitals)
3	Semantic payload success/failure rates tracked
4	Render pipeline latency measured per payload type
5	Perfect observability: end-to-end payload-to-pixel tracing, formally verified performance bounds

The Mythos Semantic UI Suite (M-SUI-S)

Traditional UI benchmarks measure Lighthouse scores. The M-SUI-S evaluates semantic separation, injection resistance, and rendering acceleration.

Suite Composition

M-SUI-S-Security

1. XSS Injection: 100+ tests injecting malicious HTML/JS via LLM payloads to verify renderer rejection.

1. Payload Fuzzing: 50+ tests sending malformed semantic payloads to verify validation and error handling.

M-SUI-S-Rendering

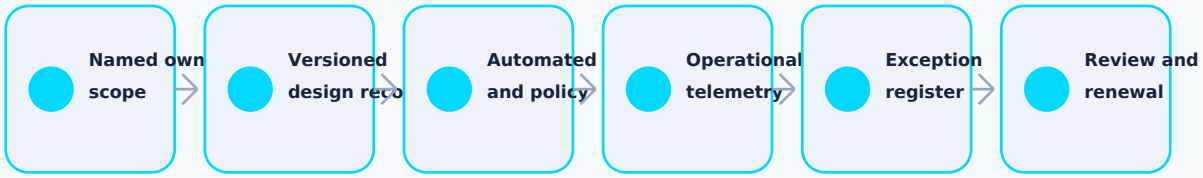
1. High-Density Load: 50+ tests rendering 1M+ data points to verify WebGPU maintains 60fps.
1. Multi-Target Fidelity: 50+ tests verifying the same payload renders correctly on Web, Desktop, and CLI.

Execution Protocol

ASSURANCE AND ADOPTION

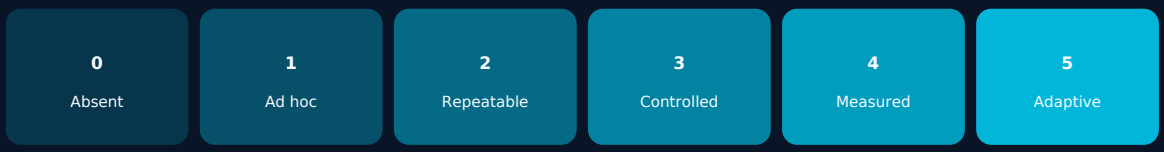
Implementation evidence and review gates

Required assurance chain



A passing score is not sufficient without current evidence. High-assurance adoption requires traceable design decisions, reproducible builds, independently reviewable tests, controlled secrets, runtime observability, failure exercises, and a dated exception register.

Assurance maturity spectrum



Current authority register

Authority	Version / snapshot	Applicability	Review by
Rust Project: The Rust Programming Language and Rust Reference	Rolling official documentation; book assumes Rust 1.90.0 or later and Rust 2024 Edition	High	2026-10-22
Go Project: The Go Programming Language Specification and Effective Go	Rolling official documentation snapshot	High	2026-10-22
Python Software Foundation: Python Language Reference, Packaging, Typing, and PEP Index	Rolling official documentation snapshot	High	2026-10-22
Tauri Programme: Tauri 2 Documentation	Tauri 2 rolling documentation snapshot	High	2026-10-22
Svelte Project: Svelte Documentation	Svelte 5 rolling documentation snapshot	High	2026-10-22
TLA+ Foundation: TLA+ Language and Tools Documentation	Rolling documentation snapshot	High	2027-01-24
OpenAPI Initiative: OpenAPI Specification	3.2.0	High	2027-01-24
OpenTelemetry Project: OpenTelemetry Documentation and Specification	Rolling specification snapshot; profiles public alpha and declarative configuration stable in 2026	High	2026-10-22

Research limitation. Official documentation establishes published interfaces and declared behavior. It does not prove the performance, security, interoperability, cost, or operational suitability of a specific implementation. Those claims require controlled testing and retained evidence.