Glyph Tokens

A typed semantic intermediate representation between language and governed action

Hayden Lindley · Glyphd Labs

2026-07-27

Contents

Glyph Tokens

Abstract

Language models operate on tokenizer tokens, but human projects operate on commitments, conditions, authorities, dependencies, prohibitions, evidence, and actions. Treating both as undifferentiated text forces every downstream system to reinterpret meaning and risks allowing a model's plausible paraphrase to silently change the project.

Glyph Tokens are a typed semantic intermediate representation between natural language and governed operations. They compile messy input into inspectable nodes for propositions, authority, modality, scope, conditions, constraints, substitutions, uncertainty, sources, and actions. The exact source remains sovereign. A Glyph Token graph is a derived interpretation with explicit compiler and evidence dependencies.

The system is designed for coordination, not mysticism. A token is not trusted because it is typed. Trust derives from source, authority, signature, project scope, compiler policy, validators, current state, and human approval where required.

1. Why text alone is insufficient

Consider:

> “Use the old landing page as a reference, but don't change the accepted logo, and only deploy after I approve the mobile version.”

This sentence contains:

- a reference source;
- a preference;
- a prohibition;
- an accepted invariant;
- a conditional sequence;
- an authority boundary;
- a deployment action;
- an approval requirement.

A prose summary may preserve most of this, but downstream tools need exact operational distinctions.

2. Token classes

A Glyph Token may represent:

```
PROPOSITION
SOURCE
EVIDENCE
AUTHORITY
MODALITY
SCOPE
CONSTRAINT
CONDITION
DEPENDENCY
SUBSTITUTION
PROHIBITION
UNCERTAINTY
OBJECTIVE
ACTION
APPROVAL
BLOCK
SATISFACTION
```

A token includes:

```
token_id
token_type
canonical_payload
source_spans[]
project_id
authority_state
confidence_or_uncertainty
dependencies[]
```

```
compiler_profile
created_at
status
```

Relations form a graph:

```
AUTHORIZED_BY
CONDITIONED_ON
BLOCKS
SATISFIES
DEPENDS_ON
DERIVED_FROM
SUBSTITUTES_FOR
CONTRADICTS
EVIDENCED_BY
SUPERSEDES
```

3. Exact source remains available

Compilation never replaces the input. Every semantic node references exact source spans or a declared inferred origin.

This supports:

- opening the original words;
- comparing interpretations;
- recompiling under a newer compiler;
- identifying unsupported nodes;
- distinguishing quote from paraphrase;
- preserving legal or design-critical wording.

An unsupported inferred token is visually and operationally different from a source-backed one.

4. Authority and modality

The same words have different consequences depending on who said them and how.

```
“I might use blue.”        → possibility
“Use blue.”                → directive, if speaker has authority
“The design uses blue.”    → observation or claim
“Codex suggests blue.”     → agent proposal
“Blue is locked.”          → accepted invariant, if supported by decision event
```

The compiler proposes modality and authority. The runtime validates them against project identity and events. A model cannot promote its own suggestion into founder authority.

5. Conditions and substitutions

A major failure class in automation is silent substitution.

```
Use provider A.
If A is unavailable, ask before using B.
Never send private source C to cloud providers.
```

Tokens make the branch explicit:

```
ACTION use_provider(A)
CONDITION unavailable(A)
APPROVAL_REQUIRED substitute(A,B)
PROHIBITION export(C, cloud)
```

A router can now reject a substitution even if the alternative appears technically convenient.

6. Uncertainty is a token

Uncertainty should not disappear inside one confidence score. A token can express:

- ambiguous referent;
- conflicting sources;
- stale fact;
- inferred relationship;
- unresolved authority;
- approximate equivalence;
- missing evidence.

Uncertainty affects routing. High-authority or high-risk unresolved tokens can trigger stronger verification, a question, or a stop. Low-risk exact transformations can proceed without a general model call.

7. Compilation pipeline

```
immutable Capture
→ source normalization
→ candidate token graph
→ schema validation
→ source-span verification
→ authority and scope overlay
→ contradiction and substitution checks
→ human review when required
→ accepted token graph
```

The compiler can be a local 4B model, a larger model, deterministic rules, or a hybrid. The protocol does not make one model permanent.

Compiler identity includes weights or service, prompt, vocabulary, schema, normalization, and version.

8. Glyph Macros

Repeated subgraphs can be named as **Glyph Macros**:

```
macro: VERIFIED_BUILD_RETURN
expands to:
  RETURN
  EVIDENCE(build_log)
  EVIDENCE(test_log)
  LIMITATIONS
  RECEIPT_REQUIRED
  HUMAN_ACCEPTANCE_PENDING
```

Macros reduce repeated structure and support cache reuse, but expansion is deterministic, versioned, bounded, and independently inspectable. Recursive depth is limited to prevent graph bombs.

9. Token lifecycle

Tokens move through states such as:

```
CAPTURED
COMPILED
PROPOSED
ACCEPTED
ACTIVE
SATISFIED
BLOCKED
SUPERSEDED
DISPUTED
TOMBSTONED
```

A satisfied token can leave the active context while remaining in the ledger. A blocked branch can remain as a tombstone so the system does not repeatedly propose it.

10. Runtime routing

A software semantic scheduler can prioritize work by:

```
authority criticality
+ unresolved conflict
+ uncertainty
+ action risk
+ evidence need
+ downstream dependency count
+ time sensitivity
```

Examples:

- accepted `BLOCK` → deterministic stop;
- verified macro → exact cache expansion;
- `VERIFY` → evidence lane;
- `NEVER_EXPORT` → local-only route;
- unresolved high-impact substitution → approval or strongest verifier;
- malformed graph → bounded repair;
- exact low-risk transform → ordinary code.

This is the software form of the TTU idea before any hardware claim.

11. Relationship to GlyphCAS

GlyphCAS can cache a compiled graph under:

- exact input content;
- semantic graph identity;
- project and authority scope;
- compiler profile;
- dependency root;
- task contract.

Exact source identity remains separate from semantic graph identity. A semantic cache hit retrieves a previously accepted compilation under the same contract; it does not pretend the input bytes were identical.

12. Relationship to SurfaceDelta and PromptCapsules

Glyph Tokens can compile into:

- PromptCapsules for model calls;
- SurfaceDeltas for visual state changes;
- Work Orders for bounded execution;
- approval sheets;
- evidence queries;
- receipt expectations.

This makes the graph a coordination layer across models and tools, not a new top-level consumer brand.

13. Threat model

Threats include:

- authority spoofing;
- source-span tampering;
- malicious macros;
- cache poisoning;
- stale anchors;
- cross-project leakage;
- validator bypass;
- recursive expansion;
- prompt injection encoded as a high-authority token;
- hidden substitution;
- replay of old approvals.

Controls include exact source hashes, compiler versioning, signatures, project scope, authority validation, macro limits, dependency invalidation, and visible approval.

14. Evaluation

A compilation benchmark should test:

- source-span accuracy;
- token-type accuracy;
- authority/modality accuracy;
- constraint retention;
- substitution detection;
- contradiction preservation;
- graph determinism under paraphrase;
- invalid graph rate;
- repair count;
- task success when consumed by a downstream runtime;
- unauthorized action rate;
- cache reuse precision.

The strongest test uses messy real founder language rather than only clean synthetic commands.

15. Current evidence state

The protocol is deeply specified across Glyph Token, GlyphCAS, Project Core, MaxFlex, Haygent, and MCE-1 materials. Related deterministic validators and typed schemas exist in donor

systems. The complete cross-product protocol remains `SPECIFIED` until an exact reference compiler, conformance suite, and accepted integration receipt are published.

16. Limitations

No semantic graph captures every nuance. Compiler errors can become more dangerous when structured. Overtyping can make conversation rigid. Vocabulary governance is difficult. Some ambiguity should remain unresolved rather than forced into a node.

Glyph Tokens are therefore a derived operational representation, not a replacement for language, source evidence, or human judgment.

17. Conclusion

Glyph Tokens provide a practical grammar for moving from conversation to governed work. They preserve the differences that ordinary text pipelines collapse: suggestion and decision, possibility and obligation, exact source and inference, allowed substitution and forbidden substitution, completion and acceptance.

The protocol's value is not that it makes meaning perfectly discrete. It is that it makes consequential interpretation inspectable, versioned, cacheable, challengeable, and enforceable.

Source register

- `GLYPH_TOKENS_MAIN_BUILD_THREAD_HANDOFF.md`
- `LM_JEFE_FULL_SYSTEM_SPEC.md`
- GlyphCAS design report
- `MCE-1-SYSTEM-CONTRACT-v0.1.md`
- `ZEKE-4B-MAXFLEX-RUNTIME-MASTER-SPEC-v0.1.md`

Revision history

- **0.1.0 — 2026-07-27:** Initial public-safe institute edition.