

# UAS/1.0 authoring guidelines

How to fill in a Unified Agent Spec so the Agent Analyzer accepts it on the first upload

*This is a companion to UAS\_1.0\_Template.json and UAS\_1.0\_Template.docx (the fillable starting points on the Converter page). It exists because most upload failures are not typos — they are one of a small, fixed set of enum values or cross-field rules the schema enforces. Every value list below is copied directly from uas-1.0.schema.json, not paraphrased, so you can match spelling exactly.*

## 1. What the analyzer actually checks

A UAS document makes specific, checkable claims about your agent: what it decides, what code implements each decision, what limits (guardrails) apply, and what authority it has. The analyzer locates the code each claim points to and checks the claim against it. Two consequences follow directly:

- **Locators are not decorative.** Every locator (file + symbol) must be real — a function or class that actually exists in the repository you submit alongside this document.
- **Numbers must match the code.** A guardrail's threshold must be the literal number or string your code actually compares against — not a rounded or approximate value.

If you don't yet know a value precisely, it's better to leave the optional field out and describe the limit in prose (in context.limitations or a decision's statement) than to guess a number the analyzer will mark as a mismatch.

## 2. The document's required shape

Every UAS needs these top-level keys. Missing any of them is an instant schema failure, not a partial score:

- uas (just {"version": "1.0"})
- agent
- capabilities
- decisions
- interfaces (with consumes and produces)
- operations (with failure\_modes and safe\_states)
- oversight (with posture, controls, escalations)
- lifecycle (with change\_log)
- references
- execution

policies, governance.risks, scenarios, acceptance\_criteria and limitations are optional — include them when you have something true to say, not to pad the document out.

### 3. Exact allowed values — the #1 source of rejected uploads

Every one of these is a closed list. A value that reads as reasonable English but isn't on the list ("purchasing", "orchestrate" as an authority, "alert" as an on\_breach) is rejected, not auto-corrected.

| Field                                     | Allowed values (exact spelling)                                                                                                                                                                                                                                                                                               |
|-------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>agent.categories (1-3)</b>             | forecasting, demand-planning, supply-planning, inventory, replenishment, procurement, supplier-management, manufacturing, quality, warehouse, logistics, transportation, order-management, returns, finance, analytics, risk, compliance, sustainability, control-tower, copilot, orchestration — or a custom x-your-category |
| <b>capability.kind</b>                    | sense, analyze, predict, recommend, execute, converse, orchestrate                                                                                                                                                                                                                                                            |
| <b>decision.authority</b>                 | observe, recommend, act-with-approval, act-with-oversight, act-autonomous                                                                                                                                                                                                                                                     |
| <b>decision.reversibility</b>             | reversible, compensable, irreversible                                                                                                                                                                                                                                                                                         |
| <b>decision.trigger.kind</b>              | event, schedule, threshold, request                                                                                                                                                                                                                                                                                           |
| <b>decision.blast_radius.scope</b>        | item, order, shipment, site, network, enterprise                                                                                                                                                                                                                                                                              |
| <b>guardrail.on_breach</b>                | block, require-approval, escalate, degrade                                                                                                                                                                                                                                                                                    |
| <b>guardrail.scope.basis</b>              | per-decision, per-period (period is required when basis is per-period)                                                                                                                                                                                                                                                        |
| <b>interface.kind (consumes/produces)</b> | api, event, file, database, stream, ui, document                                                                                                                                                                                                                                                                              |
| <b>consumed interface.if_unavailable</b>  | halt, degrade, use-stale, substitute                                                                                                                                                                                                                                                                                          |
| <b>system.role</b>                        | system-of-record, execution, data-source, notification, model-service, other                                                                                                                                                                                                                                                  |
| <b>system.direction</b>                   | read, write, read-write                                                                                                                                                                                                                                                                                                       |
| <b>system.criticality</b>                 | required, degradable, optional                                                                                                                                                                                                                                                                                                |
| <b>failure_mode.response</b>              | halt, degrade, fallback, queue, escalate, retry-bounded                                                                                                                                                                                                                                                                       |
| <b>policy.class</b>                       | invariant, constraint, preference (constraint requires an exceptions block)                                                                                                                                                                                                                                                   |
| <b>oversight.posture</b>                  | human-in-the-loop, human-on-the-loop, human-out-of-the-loop                                                                                                                                                                                                                                                                   |
| <b>risk.category</b>                      | operational, financial, safety, security, privacy, compliance, reputational, model-behavior                                                                                                                                                                                                                                   |
| <b>risk.severity</b>                      | negligible, minor, moderate, major, severe                                                                                                                                                                                                                                                                                    |
| <b>risk.likelihood</b>                    | rare, unlikely, possible, likely, almost-certain                                                                                                                                                                                                                                                                              |
| <b>lifecycle.change_log[].type</b>        | added, changed, deprecated, removed, fixed, security                                                                                                                                                                                                                                                                          |
| <b>model_dependency.update_policy</b>     | pinned, managed-upgrade, provider-continuous                                                                                                                                                                                                                                                                                  |
| <b>model_dependency.revalidation</b>      | full, affected, none                                                                                                                                                                                                                                                                                                          |

*Common mix-ups seen in real submissions:*

- "purchasing" is not a category — use procurement.

- "cold-chain" or "monitoring" are not categories — use the nearest real one (quality, risk, logistics...) or a custom x-cold-chain.
- "orchestrate" is a capability kind, not a decision authority — authorities never include it.
- "alert" is not an on\_breach value — use escalate.
- "product" is not a blast\_radius scope — the closest real value is usually item.
- "email" is not an interface kind — an email is usually a document or event, depending on what you're describing.

## 4. Rules between fields, not just within one

These four are the ones that fail silently if you only check each field on its own:

- **Acting decisions need guardrails + escalation.** If a decision's authority is act-with-oversight or act-autonomous, it MUST also have at least one guardrail and an escalation. An autonomous or oversight-level action with no stated limit is treated as missing information, not as “no limit needed.”
- **Non-reversible decisions need a reversal statement.** If a decision's reversibility is compensable or irreversible, it MUST also have a reversal field explaining how (compensable) or stating plainly that it cannot be undone (irreversible).
- **Constraint-class policies need exceptions.** A policy with class "constraint" MUST have an exceptions block (who can approve breaking it, and under what statement). invariant and preference do not need this.
- **Orchestrators need composition.** If agent.kind is "orchestrator", the document MUST include a composition block naming its member agents.
- **At least one escalation is required.** oversight.escalations needs at least one entry — even an advisory, nothing-is-executed agent needs to say who reads its output.
- **Guardrail thresholds can't be booleans.** A guardrail's threshold must be a number, a string, an array, or a quantity/money object — never a bare true/false. Express a yes/no limit as a string or a real bound instead.

## 5. Length limits worth knowing before you write

- agent.summary: 200 characters. Write it last, after the detail sections, and cut it hard.
- guardrail.parameter, escalation.to, model\_dependency.role and most other short fields: 120 characters.
- escalation.when: 200 characters.

## 6. Locators: the part a re-read of the code fixes fastest

A locator is {"file": "...", "symbol": "..."}.

Point it at the exact function, method or class that implements the capability or decision — not the file that merely imports it, and not a class when you mean one of its methods (use Class.method).

- Use the path as it appears from the repository root, exactly as it would be after a fresh clone (e.g. `src/agent/orders.py`, not an absolute path).
- A guardrail's own locator should point at the specific check in code (the if-statement or validation function), which is often a different place than the decision's own locator.
- If you genuinely can't locate something in code yet, leave the locator out rather than pointing it at the nearest plausible file — an unresolved element is reported honestly; a wrong locator is reported as a false grounding.

## 7. Formulas and guardrail numbers: write what the code computes, not what it should compute

The analyzer symbolically compares a formula's expression against the arithmetic it finds in your code slice. It can only verify formulas built from ordinary arithmetic (+ - \* / parentheses, min/max, round) over named variables — not prose, not pseudocode.

- Match operator to what the code actually does: lte for `<=`, not lt, if the code uses `<=`.
- A threshold that is compared as text (e.g. `status == "delivered"`, or `status in ["pending", "confirmed"]`) is not a numeric bound — the analyzer will correctly report it unverifiable rather than pretend to check it. That is expected, not an error to fix.
- If the code's constant differs from a rounded number you remember (25000 vs 25,000.00, 30 vs 30.0), use the code's own literal.

## 8. Be honest about authority — the analyzer checks it against real code

Declaring a decision as act-autonomous when the code in fact only prints or logs a claimed action (and never writes to a store or calls an external system) will be caught: the analyzer looks for real write side effects reachable from that decision's code and reports a mismatch if it declared more authority than the code exercises. If your agent's output describes an action in words without the code performing it, declare the authority truthfully (usually recommend) and say so plainly in `context.excluded` or `limitations` — that is a correct, verifiable document, not a weaker one.

## 9. Suggested order to fill the template in

1. agent block and `context.excluded` — what this is and what it deliberately does not do.
2. capabilities — one per distinct thing the agent senses, predicts, recommends or executes.
3. decisions — one per capability that actually branches on something; attach the real locator as you write each one.
4. `interfaces.consumes` / `produces` and systems — what data goes in and out, and where it lives.
5. `operations.failure_modes` / `safe_states` — what happens when an input is missing or invalid.
6. oversight, policies, governance.risks, lifecycle, references, execution — fill last, once the shape above is stable.

Then validate locally against `uas-1.0.schema.json` (any JSON Schema validator) before uploading — it catches every rule in sections 3-5 above instantly, without needing the repository at all.