ENTWICKLERLEITFADEN

Custom Rulesets verstehen

Für Entwickler, die eigene Regelwerke schreiben · ab Plan Project

---

Ein Entwicklerleitfaden für eigene Regeln, was ein Regelwerk ausdrücken kann, was bewusst nicht, und wie aus einem Integrationshandbuch eine Datei wird, die ein Audit übersteht.

Stand 16. August 2026. Formatversion **1.0**. Das maßgebliche JSON-Schema, `ruleset-1.0.schema.json`, liegt jeder Installation bei und ist auf Anfrage erhältlich: **support@logiq-vda5050.com**.

Was ein Custom Ruleset ist

Keine Flotte scheitert bei der Inbetriebnahme allein an VDA5050. Das Integrationshandbuch jedes Fahrzeugherstellers ist voller Sätze, die auf der Anlage bindend und für den Standard unsichtbar sind, *„pickFromBin braucht eine binId der Form BIN-0000“*, *„Visualization-Nachrichten dürfen nicht schneller als alle 200 ms kommen“*. Heute stehen diese Sätze in einem PDF, das niemand gegen den tatsächlichen Verkehr prüft.

Ein Custom Ruleset trägt diese Sätze als JSON-Datei in den Analyzer. Einmal hochgeladen, laufen seine Prüfungen in derselben Maschinerie, die auch den Standard auswertet: derselbe Lauf, derselbe Bericht, dieselbe Beweiskette vom Befund bis zur konkreten Nachricht und zum Feld, das ihn ausgelöst hat. Befunde Ihrer Regeln tragen das Familienkürzel **CR**, sind einem Fahrzeug zugeordnet und, wenn Sie es sagen, der Seite, die sie verursacht hat.

Zwei Eigenschaften des Entwurfs sind nicht verhandelbar und alles Weitere in diesem Leitfaden folgt aus ihnen:

1. **Ein Regelwerk ist Daten, niemals Code.** Die Datei ist beschreibendes JSON gegen ein veröffentlichtes Schema, gedeutet von einem geschlossenen Satz von sieben Regeltypen. Nichts darin wird jemals ausgeführt. Jeder Bericht trägt eine Integritätsaussage, dass auf einer Installation nur signierte Software läuft; eine hochgeladene Datei, die ausgeführt werden könnte, machte diese Aussage falsch.
2. **Eine eigene Regel verschiebt niemals das VDA5050-Urteil.** Der Bericht zeigt zwei Urteile nebeneinander: den Standard allein und den Standard samt Ihren Regeln. Das erste ist berechnet, bevor ein einziger eigener Befund existiert. Alles andere erlaubte einem Hersteller, seine Konformität durch eine mildere Datei zu verbessern, das Gegenteil dessen, wofür dieses Produkt da ist.

Was Regelwerke können: und was nicht

Ein Regelwerk beurteilt **eine Nachricht auf einmal** gegen Konstanten, die Sie hingeschrieben haben, plus zwei Eigenschaften je Fahrzeug über den Nachrichtenstrom (Abstand und Monotonie). Das deckt einen erstaunlichen Anteil jedes Integrationshandbuchs ab. Alles deckt es nicht ab, mit Absicht.

In Reichweite:

- Vorhandensein, Typ, exakter Wert, erlaubte Menge, Muster, Zahlenbereich oder Länge eines Feldes, je Nachricht, an jedem Pfad, auch in gefilterten Array-Elementen (`actions[?actionType=pickFromBin]`).
- Eine ganze Nachricht oder ein Teilbaum gegen ein eingebettetes JSON-Schema (Draft 2020-12), für Formen, die einem einzelnen Prädikat zu reich sind.
- Wie viele Elemente ein Array hält (`nodes[]` höchstens 64).
- Der Abstand zwischen Nachrichten einer Art, je Fahrzeug (`state` mindestens alle 1 000 ms, `visualization` nicht schneller als 200 ms).
- Die Monotonie eines Zahlenfeldes, je Fahrzeug (ein Zähler ohne Lücken, ein Wert, der nie sinken darf).
- Ein Wert, der gar nicht vorkommen darf, wahlweise nur unter einer Bedingung (*während des Ladens darf `driving` nicht true sein*).
- Ein Tor auf Nachrichtenebene (`when`), damit eine Regel nur in der Situation spricht, die das Handbuch beschreibt.

Außer Reichweite, mit Absicht:

- **Zwei Felder miteinander vergleichen.** Prädikate vergleichen gegen Konstanten in der Datei, nie gegen einen anderen Wert der Nachricht (*„velocity muss unter dem deklarierten maxSpeed bleiben“* ist nicht ausdrückbar).
- **Arithmetik und abgeleitete Werte.** Keine Summen, keine Raten, keine Differenzen.
- **Korrelation über Nachrichtenarten hinweg.** *„Jede Order muss in einer State-Nachricht bestätigt werden“* ist die Aufgabe der eingebauten Kohärenzprüfungen, ein Regelwerk verknüpft nie zwei Nachrichten.
- **Zustandsautomaten.** Jenseits der Monotonie gibt es kein Gedächtnis zwischen Nachrichten; die Prüfung von Aktions-Lebenszyklen ist eingebaut, nicht schreibbar.
- **Alles Umgebungsabhängige.** Keine Uhrzeit, kein Netzwerk, keine Nachschlagewerke, kein Zufall. Dieselbe Datei über derselben Aufzeichnung liefert dieselben Befunde, jedes Mal, auf jeder Installation.
- **Das Urteil, den Score oder die Schweregrad-Gewichte des Standards anfassen.** Eigene Befunde zählen im Custom-Rules-Urteil und nirgendwo sonst.

Passt ein Satz Ihres Handbuchs in keine der sieben Formen, ist die Antwort kein achter Regeltyp, jeder weitere Typ wäre ein Schritt in Richtung Programmiersprache und die verbietet Eigenschaft 1. Schreiben

Sie stattdessen an support@logiq-vda5050.com; verallgemeinert sich der Satz, wird er eine eingebaute Prüfung, die alle bekommen.

Aufbau der Datei

Ein Regelwerk ist ein einzelnes JSON-Objekt. Dieses ist vollständig und ließe sich hochladen:

```
{
  "ruleset": "1.0",
  "name": "Acme R-Series",
  "version": "1.2.0",
  "date": "2026-08-09",
  "vendor": "Acme Robotics",
  "source": "Acme VDA5050 Integration Guide, Rev. C",
  "target_versions": ["2.1.0"],
  "checks": [
    {
      "id": "ACME_PICK_BIN_ID",
      "group": "Actions",
      "label": "pickFromBin carries a binId",
      "description": "Acme vehicles reject pickFromBin without a binId of the form BIN-
0000.",
      "spec_ref": "Integration Guide Rev. C, 4.2.1",
      "severity": "Major",
      "addressee": "master_control",
      "applies_to": { "kind": "order" },
      "rule": {
        "type": "field",
        "path": "nodes[].actions[?actionType=pickFromBin].actionParameters[?
key=binId].value",
        "must": { "is": "string", "matches": "^BIN-[0-9]{4}$" }
      }
    }
  ]
}
```

Der Kopf identifiziert die Datei; der Analyzer verzeichnet Name, Version und den SHA-256 der Datei in jedem Bericht, der sie verwendet hat.

Feld	Pflicht	Bedeutung
`ruleset`	ja	Formatversion, immer `"1.0"`.
`name`	ja	Name des Regelwerks, bis 120 Zeichen. Erscheint überall, wo der Bericht seine Regeln nennt.
`version`	ja	Ihre Versionsangabe, bis 40 Zeichen. Versionieren Sie wie Software, der Bericht friert sie ein.
`date`	ja	`YYYY-MM-DD`, ein echtes Kalenderdatum.
`vendor`	nein	Wessen Regeln das sind.
`source`	nein	Das Dokument, dem die Regeln entnommen sind, wer den Bericht liest, will nachschlagen.
`target_versions`	ja	Für welche VDA5050-Versionen die Regeln gelten: `"2.1.0"`, `"3.0"` oder beide. Das Regelwerk wird nur angeboten, wo die Zielversion des Laufs passt.
`checks`	ja	1 bis 200 Prüfungen.

Unbekannte Schlüssel werden überall in der Datei abgelehnt, ein Tippfehler wie `"severty"` lässt den Upload scheitern, statt still ignoriert zu werden.

Jede Prüfung trägt erst ihre Papiere, dann ihre Regel:

Feld	Pflicht	Bedeutung
`id`	ja	`^[A-Z][A-Z0-9_]{2,63}$`, eindeutig in der Datei und ohne Kollision mit einer eingebauten Prüfkennung. Mit Herstellerpräfix: `ACME_PICK_BIN_ID`.
`label`	ja	Eine Zeile, bis 160 Zeichen, die Überschrift des Befunds.
`description`	ja	Was die Regel bedeutet und warum es sie gibt, bis 1 000 Zeichen. Schreiben Sie für den Inbetriebnehmer, der den Befund sieht.
`severity`	ja	`Critical`, `Major`, `Minor` oder `Info`, gezählt im Custom-Rules-Urteil.
`applies_to.kind`	ja	Die Nachrichtenart, die die Regel liest: `order`, `state`, `instantActions`, `connection`, `visualization` oder `factsheet`.
`addressee`	nein	Wer eine Verletzung verursacht hat: `master_control` oder `vehicle`. Sagen Sie es, wo das Handbuch es sagt, Zuordnung beendet Inbetriebnahme-Diskussionen.
`group`	nein	Gruppiert Prüfungen unter einer Überschrift in der Auswahl und im Bericht.
`spec_ref`	nein	Kapitel und Vers im Herstellerdokument, z. B. `"Rev. C, 4.2.1"`. Wird zur Empfehlung des Befunds, wenn Sie keine angeben.
`recommendation`	nein	Was bei einer Verletzung zu tun ist, bis 500 Zeichen.

Die Pfadsprache

Regeln zeigen mit einem Punktpfad in die Nutzlast der Nachricht. Es gibt drei Segmentformen und nur diese drei:

Segment	Bedeutung
`name`	Steigt in das Objektfeld `name` hinab.
`name[]`	Iteriert jedes Element der Liste `name`.
`name[?feld=wert]`	Iteriert die Elemente von `name`, deren `feld` gleich `wert` ist (als Text verglichen: `[?headerId=7]` trifft die Zahl 7).

Segmente verbinden sich mit Punkten: `nodes[].actions[?actionType=pickFromBin].actionParameters[?key=binId].value`. Pfade sind relativ zur Nutzlast, MQTT-Umschläge werden vor der Auswertung entfernt, schreiben Sie also nie ein `payload.`-Präfix. Benennt ein Pfad mehrere Werte (durch `[]` oder einen Filter), beurteilt die Regel **jeden davon** und jeder Befund zeigt den konkreten Fundort, `nodes[2].actions[0]…`, in derselben Notation wie jeder eingebaute Befund.

Es gibt keine Wildcards, keine numerischen Indizes und keinen „Suche-überall“-Operator. Ein Pfad, der durch ein fehlendes Feld führt, benennt schlicht nichts, weshalb Existenz ein eigenes Prädikat mit eigener Semantik ist, siehe unten.

Die sieben Regeltypen

`field` : ein Prädikat auf einem Wert

Das Arbeitspferd: jeden Wert finden, den der Pfad benennt, und jeden gegen den `must` -Block halten. Verfügbare Prädikatschlüssel, in dieser Reihenfolge geprüft:

Schlüssel	Besteht, wenn der Wert…
`exists`	vorhanden ist (siehe unten, Existenz wird am Elternobjekt beurteilt).
`is`	den Typ hat: `string` , `number` , `integer` , `boolean` , `object` , `array` . Booleans zählen nie als Zahlen.
`equals`	der Konstante exakt gleicht.
`in`	eine der aufgezählten Konstanten ist.
`matches`	ein String ist, der den regulären Ausdruck enthält, verankern Sie mit `^…$` , wenn Sie den ganzen Wert meinen.
`min` / `max`	eine Zahl im Bereich ist. Eine Nicht-Zahl fällt durch.
`min_length` / `max_length`	ein String oder eine Liste innerhalb der Längengrenzen ist.

Mehrere Schlüssel verbinden sich als *und*; der Befund nennt das erste gebrochene Versprechen in Worten: `is 'BIN-12', which does not match /^BIN-[0-9]{4}$/` .

Existenz hat eine Feinheit. `"must": { "exists": true }` muss auf einem einfachen Feldnamen enden und das *Elternobjekt* entscheidet: ein fehlender Container ist kein fehlendes Feld. Fehlt `batteryState` komplett, bleibt eine Regel auf `batteryState.acmeCellTemp` stumm, sprechen soll die andere Regel, die `batteryState` selbst prüft. So zeigt jeder Befund auf seine eigene Ursache, statt einen fremden Befund zu wiederholen.

Eine `when` -Bedingung (nächster Abschnitt) schaltet die ganze Regel je Nachricht.

`schema` : ein eingebettetes JSON-Schema

Wenn die Form für ein Prädikat zu reich ist: ein JSON-Schema (Draft 2020-12) einbetten und auf die ganze Nutzlast oder einen `path` anwenden:

```
{
  "type": "schema",
  "path": "acmeDiagnostics",
  "schema": {
    "type": "object",
    "required": ["frameCounter", "motorTemp"],
    "properties": {
      "frameCounter": { "type": "integer", "minimum": 0 },
      "motorTemp": { "type": "number", "maximum": 90 }
    }
  }
}
```

Jede Schemaverletzung wird ein eigener Befund mit angehängtem inneren Pfad. Das eingebettete Schema wird beim Upload selbst validiert, ein kaputtes Schema lässt den Upload scheitern, nicht die Analyse.

`cardinality`: wie viele

Zählt je Nachricht, wie viele Elemente der Pfad benennt und hält die Zahl gegen `min`, `max` oder beides (mindestens eines ist Pflicht):

```
{ "type": "cardinality", "path": "nodes[]", "max": 64 }
```

„Acme-Fahrzeuge puffern höchstens 64 Nodes je Order“ ist eine Zeile. Der Befund liest sich `78 present, at most 64 allowed`.

`interval`: Nachrichtenabstand, je Fahrzeug

Beurteilt die Zeit zwischen aufeinanderfolgenden Nachrichten der Art der Prüfung, je Fahrzeug, anhand der Zeitstempel der Nachrichten selbst:

```
{ "type": "interval", "min_ms": 200 }
```

`min_ms`, `max_ms` oder beides; kein `path`. Die Timing-Toleranz der Analyse gilt mit derselben Disziplin wie bei den eingebauten Intervallprüfungen: sie **weitet** das erlaubte Band nur, eine Toleranz kann also einen Grenzfall entschuldigen, aber nie eine Verletzung erzeugen. Ein systematisch daneben liegender Strom ist ein Fakt, nicht tausend Befunde, die Prüfung meldet **einen Befund je Fahrzeug und Richtung**, mit Anzahl und schlechtestem Wert: `AGV-3: 41 intervals below 200 ms, worst 12 ms`.

`sequence`: Monotonie, je Fahrzeug

Beobachtet ein Zahlenfeld über den Strom eines Fahrzeugs:

```
{ "type": "sequence", "path": "acmeDiagnostics.frameCounter", "mode": "gapless" }
```

Modus	Jeder Wert muss…
`increasing`	strikt größer als der vorige sein.
`non_decreasing`	größer oder gleich dem vorigen sein.
`gapless`	exakt der vorige plus 1 sein.

Nachrichten, in denen der Pfad keine Zahl benennt, werden übersprungen, ohne das Gedächtnis zurückzusetzen. Der Befund nennt beide Nachrichten: `is 17 after 19 (message 204), expected 20`.

`forbidden`: ein Wert, der nicht vorkommen darf

Das Gegenstück zu `field`: die Regel schlägt an, wenn der Pfad etwas benennt. Nackt verbietet sie das Vorhandensein; mit `equals`, `in` oder `matches` nur die genannten Werte:

```
{
  "id": "ACME_NO_SEMIAUTOMATIC",
  "label": "SEMIAUTOMATIC is not implemented",
  "description": "Acme R-Series firmware has no SEMIAUTOMATIC mode; a vehicle reporting it
is misconfigured.",
  "severity": "Critical",
  "addressee": "vehicle",
  "applies_to": { "kind": "state" },
  "rule": { "type": "forbidden", "path": "operatingMode", "equals": "SEMIAUTOMATIC" }
}
```

`transition`: erlaubte Übergänge eines Zustandswerts

Ein Feld, das sich nur entlang erlaubter Paare bewegen darf, je Entität, je Fahrzeug. Die Regel verfolgt jede Entität, benannt über `id_field`, durch die angegebenen `collections` im Strom eines Fahrzeugs; ändert sich ihr `value_field`, muss der Übergang eines der `allowed`-Paare `[from, to]` sein:

```
{
  "type": "transition",
  "collections": ["actionStates"],
  "id_field": "actionId",
  "value_field": "actionStatus",
  "states": ["WAITING", "RUNNING", "FINISHED", "FAILED"],
  "allowed": [["WAITING", "RUNNING"], ["RUNNING", "FINISHED"], ["RUNNING", "FAILED"]]
}
```

`states` grenzt ein, welche Änderung überhaupt beurteilt wird: Ein Wechsel in einen Wert außerhalb dieser Menge bleibt der eigenen Enum-Prüfung überlassen, ein falscher Wert wird einmal gemeldet, als falscher Wert, nicht doppelt als falscher Wert und falscher Übergang. Verglichen wird ohne Beachtung der Groß-/Kleinschreibung: VDA5050 1.1 schreibt seine Action-Status klein, erst 2.0 machte Großbuchstaben zur Regel, wie ein Wert geschrieben wird, ist Sache der Enum-Prüfung, nicht dieser Regel.

Bedingungen: `when`

`field` - und `forbidden` -Regeln nehmen einen optionalen `when` -Block, der die Regel auf Nachrichtenebene schaltet, die Regel spricht nur, wo die Situation des Handbuchs vorliegt:

```
{
  "type": "forbidden",
  "when": { "path": "batteryState.charging", "equals": true },
  "path": "driving",
  "equals": true
}
```

Während des Ladens darf das Fahrzeug nicht fahrend melden. Ein `when` enthält `path` plus eines von: nichts (am Pfad existiert etwas), `exists: false` (nichts existiert) oder `equals` / `in` / `matches` (ein beliebiger Wert am Pfad erfüllt das Prädikat). Merken Sie sich die Richtung: `when` wählt Nachrichten aus, der Filter `[?feld=wert]` wählt Array-Elemente aus, Korrelation *innerhalb* eines Array-Elements gehört in den Pfad, nicht in `when` .

Was zur Analysezeit geschieht

- Ein Regelwerk wird für einen Lauf angeboten, wenn seine `target_versions` die Zielversion des Laufs enthalten. Ausgewählt wird auf der Analyseseite und im Capture-Profil; jede Prüfung lässt sich einzeln an- und abschalten, und eine abgeschaltete Prüfung steht im Bericht als *abgeschaltet*, nie stillschweigend fehlend.
- Eine Prüfung, deren Nachrichtenart in der Aufzeichnung nie vorkommt, wird als **ohne Daten** berichtet, nicht als ein Bestehen, das niemand verdient hat.
- Befunde tragen `type: "custom"` , das CR-Kürzel, Ihren Schweregrad, das Fahrzeug, Ihren `addressee` und eine Beschreibung der Form `label: Detail` . Sie lassen sich dispositionieren und waiven wie jeder eingebaute Befund.
- Eine verschriebene Regel kann keinen Bericht fluten: nach 500 Befunden je Prüfung wird der Schnitt im letzten Befund angesagt, `… and 1 250 further occurrences were not listed individually` , nie still vollzogen.
- Der Bericht zeigt **zwei Urteile nebeneinander**: VDA5050 allein und Custom Rules. Eine eingefrorene Kopie jedes verwendeten Regelwerks liegt neben den Berichtsdateien und das Prüfprotokoll verzeichnet Name, Version und SHA-256 jedes einzelnen. Wer den Upload später ersetzt oder löscht, ändert nicht, womit ein bestehender Bericht gemessen wurde. Im signierten Report-Paket reisen die eingefrorenen Regelwerke unter Manifest und Signatur mit.

Wo eine Regel der Norm widerspricht

Eine Werksnorm *soll* strenger sein als VDA5050. „Bei uns fährt nur AUTOMATIC" verengt fünf erlaubte Betriebsarten auf eine und jede Nachricht, die das erfüllt, erfüllt auch die Norm. Genau dafür ist die Datei da und das wird nie gemeldet.

Wissenswert ist der andere Fall: eine Regel, die **keine normkonforme Nachricht je erfüllen kann**. Eine Flotte, die daran gemessen wird, kann nicht konform sein und fast nie ist das gewollt. Es ist ein Tippfehler, ein Feld, das die Norm zwischen zwei Versionen umbenannt hat, oder eine Regel zu einer anderen Spezifikation.

Die Ruleset-Seite trägt deshalb einen Abschnitt *Against the standard*. Beide Seiten sind Daten. Eine Regel ist ein Prädikat über einen Pfad, die Norm liegt als JSON Schema im Conformance Pack. Die Frage wird also entschieden und nicht geraten. Für einen Pfad akzeptiert die Regel eine Menge von Werten und die Norm eine andere:

	Bedeutung	**Meldung**
kein gemeinsamer Wert	keine konforme Nachricht besteht die Regel	**Widerspruch**
Regelmenge liegt in der Normmenge	die Werksnorm verengt	nie
die Regel lässt auch zu, was die Norm verbietet	die beiden Urteile widersprechen sich	als *wider than*

Heute entschieden: ein Wert oder Muster außerhalb einer Aufzählung, ein Typ, den die Norm anders festlegt, ein Zahlenbereich außerhalb des normierten, ein Pflichtfeld, das verboten oder als abwesend verlangt wird, eine Pflichtliste, die leer sein soll, das gleichzeitige Verbot aller erlaubten Werte eines Feldes sowie ein Aktionsübergang, den die Norm nicht vorsieht.

Je Version geprüft, denn die Versionen unterscheiden sich. Eine Regel auf `safetyState.eStop` stimmt für 2.1.0 und benennt in 3.0 nichts, weil das Feld dort `activeEmergencyStop` heißt, ein unverändert übernommenes Ruleset verstummt dort und genau das sagt der Abschnitt.

Drei Dinge werden bewusst **nicht** gemeldet. Ein eigenes Herstellerfeld zu verlangen ist kein Widerspruch: jedes `additionalProperties` in den mitgelieferten Schemata steht auf `true`, die Norm erlaubt Zusatzfelder also ausdrücklich. Eine Regel mit `when` wird nicht beurteilt, denn eine Bedingung, die nie zutrifft, macht sie gegenstandslos statt widersprüchlich und die Unterscheidung braucht den Verkehr. Und Zeitverhalten wird gar nicht beurteilt, VDA5050 nennt keine Senderate, „State alle 200 ms" kann ihr also nicht widersprechen.

Eine Datei für zwei Versionen trägt Regeln oft paarweise, je eine für jede Schreibweise eines umbenannten Feldes. Diese Form wird erkannt und bleibt stumm; ein Pfad, den **keine** Version der Datei kennt, wird gemeldet, denn das ist ein Tippfehler und die Regel greift nirgends.

Nichts davon rührt an ein Urteil. Es ist eine Eigenschaft der Datei, festgestellt beim Lesen der Datei und steht deshalb auf der Ruleset-Seite und nicht in jedem Report.

Regeln schreiben, die echten Verkehr überleben

Beginnen Sie beim Handbuch, nicht beim Verkehr. Jede Prüfung sollte ein Satz aus dem Herstellerdokument sein, mit `spec_ref` auf Kapitel und Vers. Was Sie nicht zitieren können, ist eine Meinung, keine Regel.

Schreiben Sie die Beschreibung für den Leser des Befunds, einen Inbetriebnehmer um zwei Uhr nachts: was die Regel bedeutet, warum es sie gibt, was zu tun ist. Das `label` ist die Überschrift, `recommendation` der Ausweg.

Wählen Sie Schweregrade wie der Hersteller. `Critical` : das Fahrzeug stoppt oder weist die Order rundweg ab. `Major` : die Order degradiert oder das Verhalten liegt außerhalb der Spezifikation. `Minor` : funktioniert, aber neben den deklarierten Werten. `Info` : sehenswert, nicht streitwert.

Ordnen Sie zu, wo Sie können. `addressee` macht aus *„es gibt einen Befund“* ein *„das Leitsystem hat das gesendet“*, der Unterschied zwischen einer Debatte und einer Korrektur.

Verankern Sie Ihre regulären Ausdrücke. `matches` sucht im String; `"BIN-[0-9]{4}"` akzeptiert `XBIN-12345` . Schreiben Sie `^BIN-[0-9]{4}$` , wenn Sie den ganzen Wert meinen.

Beweisen Sie jede Regel an einer echten Aufzeichnung. Dieser Schritt ist nicht optional. Eine generierte oder handgeschriebene Regel scheitert auf zwei stille Arten: sie **schlägt nie an** (ein vertippter Pfad, der Bericht liest sich dann als „alles in Ordnung“) oder sie **schlägt immer an** (ein zu scharfes Prädikat, der Kunde ertrinkt in Befunden und schaltet das Regelwerk ab). Das Lint-Werkzeug lässt ein Regelwerk trocken gegen eine Beispielaufzeichnung laufen, zählt Treffer je Prüfung und markiert beide Enden der Tabelle:

```
 # Generieren: das Integrationshandbuch gegen das Schema übersetzen
python3 tools/make_ruleset.py acme_guide.md \
    --name "Acme R-Series" --vendor "Acme Robotics" --version 1.0.0

# An einer echten Aufzeichnung beweisen
python3 tools/lint_ruleset.py acme.ruleset.json --sample capture.ndjson
```

Eine Datei geht erst in Produktion, wenn beide Gerüche gelesen und erklärt sind.

Upload, Rechte und Grenzen

Der Upload (Seitenleiste → Rulesets) prüft genau eines: dass die Datei lesbar und verstanden ist, schemagültig, jeder Pfad parsebar, jeder reguläre Ausdruck übersetzbar, jedes eingebettete Schema gültig, jede Kennung eindeutig und ohne Kollision mit einer eingebauten Prüfung. Eine Datei, die das besteht, kann eine Analyse später nicht überraschen. Ob ihre Regeln *klug* sind, ist Ihre Sache und der Lint-Schritt oben ist der Ort, an dem sich das entscheidet.

Grenze	Wert
Pläne	Ab **Team** (5 Regelwerke); unbegrenzt in **Enterprise** und auf einer dedizierten Installation
Wer hochladen darf	Superuser, ein Regelwerk ändert, womit die Läufe aller Mitglieder bewertet werden
Prüfungen je Datei	200
Dateigröße	2 MB
Befunde je Prüfung und Lauf	500, Schnitt wird im Bericht angesagt
Prüfkennung	`^[A-Z][A-Z0-9_]{2,63}$`, Herstellerpräfix empfohlen
Geltung	Nicht projektgebunden: ein Regelwerk beschreibt einen Hersteller und wird überall angeboten, wo die Zielversion passt

Fragen zu Custom Rulesets: **support@logiq-vda5050.com** LogistiXpert · www.logistixpert.de