ENTWICKLERLEITFADEN

Custom Rulesets verstehen

Für Entwickler, die eigene Regelwerke schreiben · ab Plan Project

Ein Leitfaden für eigene Prüfungen: was ein Regelwerk ausdrücken kann, was bewusst nicht und wie aus einem Integrationshandbuch eine Datei wird, die einem Audit standhält.

Stand 25. Oktober 2026. Formatversion **1.2**, Dateien in 1.0 und 1.1 werden weiter gelesen. Die maßgeblichen JSON-Schemas (`ruleset-1.0.schema.json` bis `ruleset-1.2.schema.json`) liegen jeder Installation bei. Sie erhalten sie auch über **support@logiq-fleets.com**. Die Formatdokumente regelwerk-format-1.1.md und regelwerk-format-1.2.md beschreiben jede Erweiterung im Einzelnen.

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 gelten und die der Standard nicht sieht. Etwa *„pickFromBin braucht eine binId der Form BIN-0000“* oder *„Visualization-Nachrichten dürfen nicht schneller als alle 200 ms kommen“*. Meist stehen diese Sätze in einem PDF, das niemand gegen den Verkehr hält.

Ein Custom Ruleset bringt sie als JSON-Datei in den Analyzer. Seine Prüfungen laufen in derselben Maschinerie, die den Standard beurteilt. Derselbe Lauf, derselbe Bericht, dieselbe Spur vom Befund bis zur Nachricht und zum Feld, das ihn ausgelöst hat. Befunde Ihrer Regeln tragen das Kürzel **CR** und nennen das Fahrzeug. Geben Sie an, welche Seite eine Verletzung verursacht, nennen sie auch diese.

Zwei Eigenschaften tragen den Entwurf. Der Rest dieses Leitfadens folgt aus ihnen.

Ein Regelwerk ist Daten, niemals Code. Die Datei ist beschreibendes JSON gegen ein veröffentlichtes Schema, gelesen von einem geschlossenen Satz von Regeltypen. Nichts darin wird je ausgeführt. Das Siegel der Anwendung sagt, dass auf einer Installation nur von uns signierte Software läuft. Eine hochgeladene Datei, die ausgeführt werden könnte, machte diese Aussage falsch.

Eine eigene Regel bewegt nie die VDA5050-Bewertung. Der Bericht zeigt die Bewertung des Standards und das *Custom Rules Verdict* als getrennte Karten. Die erste steht fest, bevor es einen einzigen eigenen Befund gibt. Sonst könnte ein Hersteller seine Konformität verbessern, indem er eine mildere Datei hochlädt.

Was ein Regelwerk kann und was nicht

Die meisten Regeln beurteilen **eine Nachricht nach der anderen** gegen Werte, die in der Datei stehen. Einige verfolgen den Strom eines Fahrzeugs: den Abstand der Nachrichten, den Verlauf einer Zahl, die Übergänge eines Zustandsfelds. Ein Regeltyp verbindet eine Nachricht mit der Antwort, die auf sie folgen muss. Kataloge führen die

Aktionen, Fehler und eigenen Felder des Herstellers auf und der Analyzer leitet feste Prüfungen daraus ab. Das deckt einen großen Teil jedes Integrationshandbuchs ab. Alles deckt es bewusst nicht ab.

In Reichweite:

- Vorhandensein, Typ, genauer Wert, erlaubte Menge, Muster, Zahlenbereich oder Länge eines Feldes, an jedem Pfad, auch in gefilterten Listenelementen (`actions[?actionType=pickFromBin]`).
- Eine ganze Nachricht oder ein Teilbaum gegen ein eingebettetes JSON-Schema (Draft 2020-12), für Formen, die ein einzelnes Prädikat nicht fasst.
- Wie viele Elemente eine Liste hält (`nodes[]` höchstens 64).
- Der Abstand zwischen Nachrichten einer Art, je Fahrzeug (`state` mindestens alle 1.000 ms, `visualization` nicht schneller als alle 200 ms).
- Wie sich eine Zahl entwickelt, je Fahrzeug oder je Wert eines Feldes, das Sie nennen: steigend, nie fallend, ohne Lücke oder ohne Wiederholung.
- Ein Wert, der gar nicht vorkommen darf oder nur in einer bestimmten Lage nicht (*beim Laden darf* `driving` *nicht true sein*).
- Die Übergänge, die ein Zustandsfeld je Objekt machen darf (`WAITING` → `RUNNING` , nie `FINISHED` → `RUNNING`).
- Eine Antwort innerhalb einer Frist: *nach einem stateRequest ein State binnen einer Sekunde*. *Meldet pick FINISHED, ändern sich die Lasten binnen zwei Sekunden.*
- Die Tabellen des Herstellers für Aktionen, Fehler und Zusatzfelder, mit Parametern, Orten, Stufen und Dauern.
- Eine Bedingung an die Nachricht (`when`), damit eine Regel nur in der Lage spricht, die das Handbuch beschreibt.
- Fristen und Grenzen als benannte Parameter mit Standardwert. Eine Anforderungsdatei kann sie für ihre Läufe setzen.

Außer Reichweite, mit Absicht:

- **Zwei Felder miteinander vergleichen.** Ein Prädikat vergleicht mit einem Wert aus der Datei, nie mit einem anderen Wert der Nachricht. *„velocity bleibt unter der deklarierten maxSpeed“* lässt sich nicht schreiben.
- **Rechnen.** Keine Summen, Raten oder Differenzen.
- **Nachrichten über Auslöser und Antwort hinaus verknüpfen.** Eine Regel vom Typ `response` verbindet zwei Nachrichten eines Fahrzeugs. Weitere Fragen wie *„jeder Auftrag wird in einem State bestätigt“* sind Sache der eingebauten Kohärenzprüfungen.
- **Alles außerhalb des Mitschnitts.** Keine Uhr, kein Netz, keine Nachschlagetabellen, kein Zufall. Dieselbe Datei über demselben Mitschnitt liefert auf jeder Installation dieselben Befunde.
- **Bewertung, Score oder Gewichte des Standards.** Eigene Befunde zählen im Custom Rules Verdict und nirgends sonst.

Passt ein Satz Ihres Handbuchs in keinen der Regeltypen, schreiben Sie an support@logiq-fleets.com. Neue Typen kommen mit einer neuen Formatversion und bleiben beschreibend. Ein Regeltyp, der rechnen könnte, wäre ein Schritt zu einer Programmiersprache. Die schließt die erste Eigenschaft aus. Ein Satz, der für jedes Fahrzeug gilt, kann auch eine eingebaute Prüfung werden, die alle bekommen.

Aufbau der Datei

Ein Regelwerk ist ein einzelnes JSON-Objekt. Dieses hier ist vollständig und besteht den Upload:

```
{
  "ruleset": "1.2",
  "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 weist die Datei aus. Jeder Bericht, der sie verwendet hat, hält ihren Namen, ihre Version und ihren SHA-256 fest.

Feld	Pflicht	Bedeutung
`ruleset`	ja	Formatversion: `"1.0"`, `"1.1"` oder `"1.2"`. Die Version bestimmt, was die Datei enthalten darf.
`name`	ja	Name des Regelwerks, bis 120 Zeichen.
`version`	ja	Ihre Versionsangabe, bis 40 Zeichen. Versionieren Sie wie Software: Der Bericht friert sie ein.
`date`	ja	`JJJJ-MM-TT`, ein echtes Kalenderdatum.
`vendor`	nein	Wessen Regeln das sind, bis 120 Zeichen.
`source`	nein	Das Dokument, aus dem die Regeln stammen, bis 300 Zeichen. Wer den Bericht liest, will nachschlagen.
`target_versions`	ja	Die VDA5050-Versionen, für die die Regeln geschrieben sind: beliebige aus `"1.1.0"`, `"2.0.0"`, `"2.1.0"` und `"3.0"`.
`checks`	ja in 1.0	Bis 200 Prüfungen. Ab 1.1 darf eine Datei auch nur Kataloge enthalten.
`catalogues`	nein, ab 1.1	Tabellen der Aktionen, Fehler und Zusatzfelder des Herstellers.
`parameters`	nein, ab 1.2	Benannte Zahlen mit Standardwert, die an Stelle einer Frist oder Grenze stehen.

Unbekannte Schlüssel werden überall in der Datei abgewiesen. Ein Tippfehler wie `"severty"` lässt den Upload scheitern. Übergangen wird er nicht.

Jede Prüfung bringt ihre Beschreibung mit, bevor sie eine Regel bringt:

Feld	Pflicht	Bedeutung
`id`	ja	`^[A-Z][A-Z0-9_]{2,63}$` , eindeutig in der Datei und verschieden von jeder eingebauten Prüfung. 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 sie für den Inbetriebnehmer, der den Befund sieht.
`severity`	ja	`Critical` , `Major` , `Minor` oder `Info` , ab 1.2 auch `Warning` . Gezählt im Custom Rules Verdict.
`applies_to.kind`	ja	Die Nachrichtenart, die die Regel liest: `order` , `state` , `instantActions` , `connection` , `visualization` oder `factsheet` .
`addressee`	nein	Wer eine Verletzung verursacht: `master_control` oder `vehicle` . Angeben, wenn das Handbuch es sagt. Die Zuordnung beendet Streit bei der Inbetriebnahme.
`group`	nein	Eine Überschrift für die Prüfung in der Auswahl und im Bericht, bis 80 Zeichen.
`spec_ref`	nein	Kapitel und Absatz im Dokument des Herstellers, z. B. `"Rev. C, 4.2.1"` . Ohne eigene Empfehlung wird daraus die Empfehlung des Befunds.
`recommendation`	nein	Was bei einer Verletzung zu tun ist, bis 500 Zeichen.

Die Pfadsprache

Eine Regel zeigt mit einem Punktpfad in die Nutzlast der Nachricht. Ein Pfad besteht aus diesen Segmenten:

Segment	Bedeutung
`name`	In das Objektfeld `name` gehen.
`name[]`	Jedes Element der Liste `name` nehmen.
`name[?field=value]`	Die Elemente von `name` nehmen, deren `field` gleich `value` ist, als Text verglichen: `[?headerId=7]` trifft die Zahl 7.

Punkte verbinden die Segmente: `nodes[].actions[?actionType=pickFromBin].actionParameters[?key=binId].value` . Pfade beginnen an der Nutzlast. MQTT-Umschläge werden vorher ausgepackt, ein Pfad beginnt also nie mit `payload.` . Nennt ein Pfad mehrere Werte, beurteilt die Regel jeden davon. Jeder Befund zeigt die genaue Stelle, `nodes[2].actions[0]…` , in der Schreibweise der eingebauten Befunde.

Es gibt keine Platzhalter, keine Zahlenindizes und keine Suche über die ganze Nachricht. Ein Pfad durch ein fehlendes Feld nennt nichts. Deshalb ist das Vorhandensein ein eigenes Prädikat mit einer eigenen Regel.

Die Regeltypen

`field` : ein Prädikat auf einem Wert

Das Arbeitspferd. Die Regel sucht jeden Wert, den der Pfad nennt, und prüft ihn gegen den Block `must` :

Schlüssel	Besteht, wenn der Wert …
`exists`	vorhanden ist. Beurteilt am Elternobjekt, siehe unten.
`is`	den Typ `string` , `number` , `integer` , `boolean` , `object` oder `array` hat. Ein Boolean zählt nie als Zahl.
`equals`	gleich dem angegebenen Wert ist.
`in`	einer der aufgezählten Werte ist.
`matches`	ein Text ist, der den regulären Ausdruck enthält. Mit `^…$` verankern, wenn der ganze Wert gemeint ist.
`min` / `max`	eine Zahl im Bereich ist. Alles andere besteht nicht.
`min_length` / `max_length`	ein Text oder eine Liste innerhalb der Längengrenzen ist.

Mehrere Schlüssel müssen alle gelten. Sie werden in der Reihenfolge der Tabelle geprüft. Der Befund nennt den ersten verletzten in Worten: `is 'BIN-12', which does not match /^BIN-[0-9]{4}$/` .

Das Vorhandensein entscheidet das Elternobjekt. `"must": { "exists": true }` muss auf einen einfachen Feldnamen enden. Ein fehlender Behälter ist kein fehlendes Feld. Fehlt `batteryState` ganz, schweigt eine Regel auf `batteryState.acmeCellTemp` . Sprechen soll die Regel auf `batteryState` selbst. So zeigt jeder Befund auf seine eigene Ursache und wiederholt nicht den einer anderen Regel.

`schema` : ein eingebettetes JSON-Schema

Ist die Form für ein einzelnes Prädikat zu reich, betten Sie ein JSON-Schema (Draft 2020-12) ein. Es gilt für die ganze Nutzlast oder für einen `path` :

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

Jede Verletzung wird ein eigener Befund, mit angehängtem innerem Pfad. Der Upload prüft das eingebettete Schema. Ein kaputtes Schema lässt den Upload scheitern, nie die Analyse.

`cardinality` : wie viele

Zählt in jeder Nachricht die Elemente, die der Pfad nennt, und hält die Zahl gegen `min` , `max` oder beide:

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

„Acme-Fahrzeuge puffern höchstens 64 Knoten je Auftrag“ ist eine Zeile. Der Befund lautet `78 present, at most 64 allowed` .

`interval` : Abstand zwischen Nachrichten, je Fahrzeug

Misst die Zeit zwischen aufeinanderfolgenden Nachrichten der Art der Prüfung, je Fahrzeug, an den Zeitstempeln der Nachrichten selbst:

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

Angegeben werden `min_ms` , `max_ms` oder beide, ein `path` entfällt. Die Zeittoleranz der Analyse **weitet** das erlaubte Band nur, wie bei den eingebauten Intervallprüfungen. Sie kann einen Grenzfall entschuldigen und erzeugt nie eine Verletzung. Ein Strom, der durchgehend abweicht, ist eine Tatsache und nicht tausend Befunde. Darum meldet die Regel je Fahrzeug höchstens einen Befund für zu kurze und einen für zu lange Abstände, mit Anzahl und schlechtestem Wert: `AGV-3: 41 intervals below 200 ms, worst 12 ms` .

`sequence` : wie sich eine Zahl entwickelt

Verfolgt ein Zahlenfeld durch den Strom eines Fahrzeugs:

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

Modus	Jeder Wert muss …
`increasing`	größer sein als der vorige.
`non_decreasing`	mindestens so groß sein wie der vorige.
`gapless`	genau der vorige plus 1 sein.
`no_gap`	höchstens der vorige plus 1 sein. Wiederholung und Rückgang sind erlaubt, ein Sprung nach vorn nicht.
`distinct`	im Strom neu sein, gleich wie weit der erste Auftritt zurückliegt.

`group_by` teilt den Strom weiter auf. Mit `"group_by": "orderId"` verfolgt die Regel `orderUpdateId` je Auftrag statt je Fahrzeug. Eine Nachricht, in der der Pfad keine Zahl nennt, wird übersprungen. Die Regel behält ihr Gedächtnis. Der Befund nennt beide Nachrichten: `is 17 after 19 (message 204), expected 20` .

Zwei Optionen sagen mehr darüber, was das Feld ist.

`counter` kennzeichnet es als Zähler der Nachrichten, die auf einem Strom gesendet werden, wie VDA5050 es von `headerId` sagt:

```
{ "type": "sequence", "path": "headerId", "mode": "non_decreasing",
  "counter": { "per": "topic", "wraps_after": 4294967295, "restart_severity": "Minor" } }
```

Eine Nachricht, die der vorigen gleicht, ist eine doppelt zugestellte Nachricht und wird nicht beurteilt. Ein neu begonnener Zähler wird einmal gemeldet, dort, wo er zurückgesprungen ist. `per` sagt, was ein Strom ist: `vehicle` (die Vorgabe) oder `topic`. `wraps_after` ist der größte Wert, den das Feld fasst; der Schritt von dort auf 0 setzt die Zählung fort. `restart_severity` ist die Stufe dieses einen Befunds, wenn die nächste Nachricht vom neuen Wert aus um eins weiterzählt. Jeder andere Rücksprung behält die Stufe der Prüfung.

`resend` gehört zu `distinct`. Die Option lässt einen Wert noch einmal zu, wenn die Nachricht dasselbe sagt wie die, die ihn zuletzt trug:

```
{ "type": "sequence", "path": "orderUpdateId", "mode": "distinct", "group_by": "orderId",
  "resend": { "ignore": ["headerId", "timestamp"] } }
```

`ignore` nennt die Felder der obersten Ebene, die beim Vergleich außen vor bleiben. Derselbe Wert über anderem Inhalt wird wie bisher gemeldet.

`forbidden`: ein Wert, der nicht vorkommen darf

Die Umkehrung von `field`: Die Regel schlägt an, wenn der Pfad etwas nennt. Allein verbietet sie das Vorhandensein. Mit `equals`, `in` oder `matches` verbietet sie nur diese 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`: die Übergänge eines Zustandsfelds

Die Regel verfolgt jedes Objekt, benannt durch `id_field`, durch die aufgeführten `collections` im Strom eines Fahrzeugs. Ändert sich sein `value_field`, muss der Übergang eines der erlaubten Paare `[from, to]` aus `allowed` sein:

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

`states` begrenzt, welche Übergänge beurteilt werden. Ein Wechsel in einen Wert außerhalb dieser Liste bleibt einer Aufzählungsprüfung überlassen. So wird ein falscher Wert einmal gemeldet, als falscher Wert. Als falscher Übergang erscheint er nicht noch einmal. Groß- und Kleinschreibung zählen beim Vergleich nicht. VDA5050 1.1 schreibt seine Aktionszustände klein, erst 2.0 machte Großschreibung verbindlich. Wie ein Wert geschrieben wird, prüft die Aufzählungsprüfung.

`response`: eine Antwort innerhalb einer Frist (ab 1.1)

Eine Nachricht (der **Auslöser**) verlangt eine zweite Nachricht (die **Antwort**) desselben Fahrzeugs innerhalb einer Frist. Wo nötig, verbindet ein Feld die beiden:

```
{
  "type": "response",
  "trigger": { "kind": "order", "path": "nodes[].actions[?actionType=liftUp]" },
  "expect": { "kind": "state", "path": "actionStates[?actionStatus=RUNNING]" },
  "correlate": [ { "trigger": "actionId", "expect": "actionId" } ],
  "within_ms": 5000,
  "from": "released"
}
```

`trigger` und `expect` nennen je eine Nachrichtenart und auf Wunsch einen `path` und ein `when`. Jedes Element unter dem Pfad des Auslösers ist ein eigener Auslöser. `not_before_ms` setzt einen frühesten Zeitpunkt für die Antwort. Bei Aktionen in Aufträgen bestimmt `from`, wann die Uhr läuft: beim Senden des Auftrags (`sent`, Standard), bei der Freigabe des Knotens (`released`) oder wenn das Fahrzeug ihn erreicht (`reached`). Die Uhr ist die Ankunftszeit im Mitschnitt, denn die Uhren von Leitsteuerung und Fahrzeug gehen verschieden. `clock: "message"` liest stattdessen die Zeitstempel, für ein Handbuch, das sie meint.

Kommt keine Antwort rechtzeitig, steht ein Befund am Auslöser, mit der gemessenen Zeit oder dem Hinweis, dass keine Antwort kam. Endet der Mitschnitt vor der Frist, sagt die Regel als Info „not judged“. Sie zählt dann weder als bestanden noch als verfehlt. Ein Auftragsupdate, das seinen Anschlussknoten erneut sendet, ist kein zweiter Auslöser.

Format 1.2 bringt zwei Schlüssel für Sätze wie *„nach dem pick steht die Last im State“*:

- `"on": "becomes"` im Auslöser: Ein Element löst nur in der Nachricht aus, in der es neu zutrifft. Spätere States, die weiter FINISHED melden, lösen nicht erneut aus.
- `"changed": "loads"` in `expect`: Eine Nachricht antwortet erst, wenn der Wert an diesem Pfad anders ist als in der letzten Nachricht des Fahrzeugs vor dem Auslöser. Die Nachricht des Auslösers selbst kann die Änderung schon tragen.

Kataloge: Aktionen, Fehler und Zusatzfelder (ab 1.1)

Der Abschnitt `catalogues` steht neben `checks`. Er enthält Tabellen, aus denen der Analyzer feste Prüfungen ableitet. Ihre Kennungen bildet er aus `id_prefix`, etwa `ACME_ACTION_PARAMETER`. Abgeleitet wird eine Prüfung nur, wo der Katalog etwas sagt, das sie beurteilen kann.

Katalog	Was er aufführt	Abgeleitete Prüfungen
`actions`	Aktionstyp, Orte (`NODE` , `EDGE` , `INSTANT`), Blockierarten, Parameter mit `required` und `must` , `max_duration_ms` , `failed_needs_result`	unbekannte Aktion (nur unter `strict`), Parameter, Ort, Blockierart, Dauer, Ergebnis, Abgleich mit dem Factsheet (Info)
`errors`	Fehlerart, Stufen, verlangte Referenzen, ein Muster für die Beschreibung, `max_duration_ms` , eine Bedingung an den State (`only_when`)	unbekannter Fehler (nur unter `strict`), Stufe, Referenzen, Beschreibung, Dauer, Kontext
`extensions`	eigene Felder des Herstellers, je Nachrichtenart, mit `must`	Wert, unbekanntes Feld (nur unter `strict`)

Eine Aktion wird je Auftrag einmal beurteilt, wie oft Updates ihren Knoten auch erneut senden. Ein Fehler wird je Episode einmal beurteilt. Die vordefinierten Aktionen des Standards wie `pick` und `startCharging` beurteilt der Analyzer ohnehin. Stehen sie im Katalog, gelten seine Einträge zusätzlich. `severity` setzt den Standard-Schweregrad je Aspekt.

Ab 1.2 bestimmt `judge` im Aktionskatalog, welche Seite er prüft. `vehicle` behält nur Dauer und Ergebnis, die das Fahrzeug meldet. `controller` behält nur, was Aufträge und Sofortaktionen senden. `both` ist der Standard. Auf einem Prüfstand sendet der Prüfstand die Aufträge, ein Fahrzeughersteller will dort meist `vehicle` . `"factsheet": false` lässt den Abgleich mit dem Factsheet weg.

Parameter (ab 1.2)

Eine Datei erklärt ihre Fristen und Grenzen einmal, mit Standardwert, und nutzt sie über den Namen:

```
"parameters": {
  "charge_start_ms": { "default": 60000, "unit": "ms", "assumed": true,
                       "description": "How soon charging starts. Not in the manual." }
}
```

`{ "param": "charge_start_ms" }` steht dann an Stelle der Zahl: in `min` / `max` eines Prädikats oder einer Kardinalität, in `min_ms` / `max_ms` eines Intervalls, in `within_ms` / `not_before_ms` einer Antwort und in `max_duration_ms` der Kataloge. `assumed: true` sagt, dass der Wert nicht vom Hersteller stammt. Der Bericht sagt das, solange niemand den Wert gesetzt hat.

Eine Anforderungsdatei, die das Regelwerk nennt, kann die Werte für ihre Läufe setzen: `{ "name": "Acme R-Series", "parameters": { "charge_start_ms": 45000 } }` . Ein Wert für einen Parameter, den das Regelwerk nicht erklärt, weist den Lauf ab. Setzen zwei Anforderungsdateien in einem Lauf denselben Wert verschieden, wird der Lauf ebenfalls abgewiesen. Die Meldung nennt beide Dateien. Der Fingerabdruck des Regelwerks ändert sich nicht, eine Festlegung über SHA-256 gilt weiter.

Bedingungen: `when`

Regeln vom Typ `field` und `forbidden` nehmen einen optionalen Block `when`, ebenso Auslöser und Antwort einer `response`. Die Regel spricht dann nur in Nachrichten, in denen die Lage des Handbuchs gegeben ist:

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

Beim Laden darf das Fahrzeug nicht melden, dass es fährt. Ein `when` enthält einen `path` und höchstens einen Test. Ohne Test verlangt es, dass am Pfad etwas steht. `exists: false` verlangt, dass dort nichts steht. Die übrigen Tests sind `equals`, `in` und `matches`. Zu unterscheiden: `when` wählt Nachrichten aus, der Filter `[?field=value]` wählt Elemente einer Liste. Was innerhalb eines Listenelements zusammengehört, gehört in den Pfad.

Ein Regelwerk hochladen

Regelwerke liegen unter **Custom Rulesets** im Analyzer-Menü. Der Eintrag erscheint, wenn der Plan das Merkmal trägt. Hochladen dürfen Superuser und Administratoren, denn ein Regelwerk ändert, woran die Läufe aller Mitglieder gemessen werden.

Der Upload prüft, ob die Datei lesbar und verstanden ist. Sie muss gegen das Schema ihrer Formatversion gültig sein. Jeder Pfad muss lesbar sein, jeder reguläre Ausdruck übersetzbar und jedes eingebettete Schema gültig. Jede Kennung muss eindeutig sein und darf keiner eingebauten Prüfung gehören. Ein benutzter Parameter muss erklärt sein. Eine Datei, die das besteht, überrascht später keine Analyse. Ob ihre Regeln *klug* sind, klärt sich an anderer Stelle, siehe unten.

Der Upload weist auch eine Datei ab, die schon gespeichert ist. Eine andere Datei unter einem Namen und einer Version, die schon vorhanden sind, wird ebenfalls abgewiesen, denn ein Bericht nennt, woran er gemessen wurde. Erhöhen Sie die Version oder löschen Sie die alte Datei.

Gleich nach dem Upload liest die Seite die Datei gegen den Standard (nächster Abschnitt). Die Liste der Regelwerke zeigt das Ergebnis in der Spalte *Standard*. Ein Regelwerk lässt sich einem Projekt zuordnen. Geöffnet zeigt es jede Prüfung mit Erklärung, Schweregrad und Quellenangabe, die Datei selbst lesbar formatiert, ihren Fingerabdruck und einen Download.

Propose from a capture auf der Seite der Regelwerke entwirft eine Datei aus einem sauberen Mitschnitt des Fahrzeugs. Sie enthält die gesendeten Aktionen mit Orten, Blockierarten, Parametern und Dauern, die gemeldeten Fehler und die Antwortzeiten auf `stateRequest` und `factsheetRequest`. Dauern und Antwortzeiten stehen beim Doppelten des längsten gesehenen Werts, aufgerundet. Nichts ist `strict`. Den Entwurf laden Sie herunter, korrigieren ihn und laden ihn hoch wie jede andere Datei. Vorher wird nichts gespeichert.

Wo eine Regel dem Standard widerspricht

Ein Werksstandard *soll* strenger sein als VDA5050. „Hier fährt nur AUTOMATIC“ engt fünf erlaubte Betriebsarten auf eine ein. Jede Nachricht, die das erfüllt, erfüllt auch den Standard. Dafür ist die Datei da. Gemeldet wird das nie.

Wissenswert ist der andere Fall: eine Regel, die **keine konforme Nachricht je erfüllen kann**. Eine Flotte, die an ihr gemessen wird, kann nicht konform sein. Das meint fast nie jemand. Es ist ein Tippfehler, ein Feld, das der Standard zwischen zwei Versionen umbenannt hat, oder eine Regel, die gegen eine andere Spezifikation geschrieben wurde.

Die Seite des Regelwerks hat deshalb einen Abschnitt *Against the standard*. Beide Seiten sind Daten. Eine Regel ist ein Prädikat auf einem Pfad und der Standard ist das JSON-Schema im Konformitätspaket. Die Frage wird also entschieden und nicht geschätzt. Für einen Pfad lässt die Regel eine Menge von Werten zu und der Standard eine andere:

	Bedeutung	Gemeldet
Kein gemeinsamer Wert	keine konforme Nachricht kann die Regel bestehen	**Widerspruch**
Die Menge der Regel liegt in der des Standards	der Werksstandard engt ein	nie
Die Regel lässt auch zu, was der Standard ablehnt	die beiden Bewertungen können auseinandergehen	als Hinweis

Heute entschieden:

- ein Wert oder Muster außerhalb einer Aufzählung;
- ein Typ, den der Standard anders festlegt;
- ein Bereich außerhalb dessen des Standards;
- ein Pflichtfeld, das verboten oder als fehlend verlangt wird;
- eine Pflichtliste, die leer verlangt wird;
- alle erlaubten Werte eines Feldes auf einmal verboten;
- ein Aktionsübergang, den der Standard nicht zulässt.

Je Version beurteilt, denn die Versionen unterscheiden sich. Eine Regel auf `safetyState.eStop` stimmt für 2.1.0 und nennt in 3.0 nichts, weil 3.0 das Feld in `activeEmergencyStop` umbenannt hat. Ein unverändert übernommenes Regelwerk verstummt dort. Der Abschnitt sagt genau das.

Einige Fälle bleiben bewusst außen vor. Eine Regel, die ein eigenes Feld verlangt, ist kein Widerspruch: Jedes `additionalProperties` in den mitgelieferten Schemas ist `true`, der Standard erlaubt zusätzliche Felder. Eine Regel mit `when` wird nicht beurteilt. Eine Bedingung, die nie eintritt, macht sie leer statt widersprüchlich. Was davon zutrifft, zeigt erst der Verkehr. Zeitverhalten wird gar nicht beurteilt. VDA5050 nennt keine Nachrichtenrate, also widerspricht *State alle 200 ms* nichts.

Eine Datei für zwei Versionen trägt oft Regelpaare, eine für jede Schreibweise eines umbenannten Feldes. Dieses Muster wird erkannt und bleibt still. Ein Pfad, den **keine** Version der Datei kennt, wird gemeldet, denn er ist ein Tippfehler und die Regel schlägt nirgends an.

Nichts davon weist die Datei ab und nichts davon berührt eine Bewertung. Ob eine Abweichung gewollt ist, entscheiden Sie. Der Befund ist eine Eigenschaft der Datei. Darum steht er auf der Seite des Regelwerks und nicht in jedem Bericht.

Regelwerke für einen Lauf wählen

Die Analyze-Seite und das Capture-Profil bieten die Regelwerke an, die passen:

- **Das Projekt des Mitschnitts.** Einem Mitschnitt in einem Projekt werden die Regelwerke dieses Projekts angeboten, wie bei Layouts und Factsheets. Einem Mitschnitt ohne Projekt werden die Regelwerke ohne Projekt angeboten.
- **Die Version des Laufs.** Ein Regelwerk beurteilt die Versionen, die es nennt. Enthält ein Mitschnitt zwei Versionen, beurteilt jedes Regelwerk die, für die es geschrieben ist. Aus dem anderen Bericht bleibt es heraus. Die Zusammenfassung des Berichts nennt, was herausblieb.
- **Jede Prüfung einzeln.** Jede Prüfung eines gewählten Regelwerks lässt sich abschalten. Der Bericht führt sie als *Switched off*, nie als fehlend.

Ein Capture-Profil behält ein Regelwerk unabhängig von seiner Version und wendet es bei der Analyse an, denn ein Mitschnitt beurteilt jede Version, die er hört.

Was der Bericht zeigt

- Das **Custom Rules Verdict** steht als eigene Karte neben der Bewertung des Standards, mit eigenen Zählern je Schweregrad und den verwendeten Regelwerken. Unter jedem Regelwerk hat jeder Parameter eine Zeile mit seinem Wert und seiner Herkunft: Standardwert, Annahme oder die Anforderungsdatei, die ihn gesetzt hat.
- Die Tabelle **Custom Ruleset Checks** führt jede Prüfung mit Ergebnis, Befunden, *Switched off* oder *No matching messages*. Eine Prüfung, deren Lage nie eintrat, hat keine Daten und zählt nie als bestanden.
- Befunde tragen das Kürzel CR, Ihren Schweregrad, das Fahrzeug und Ihren Adressaten. Sie lassen sich bewerten und als Ausnahme freigeben wie eingebaute Befunde.
- Eine falsch geschriebene Regel kann keinen Bericht fluten. Nach 500 Befunden je Prüfung sagt der letzte Befund das: *… and 1,250 further occurrences beyond the first 500 were not listed individually.*
- Jeder Bericht behält eine eingefrorene Kopie der Regelwerke, mit denen er beurteilt wurde. Der Audit-Block hält Name, Version und SHA-256 jedes Regelwerks fest, dazu die Werte der Parameter. Ein späteres Ersetzen oder Löschen der hochgeladenen Datei ändert nie, wogegen ein bestehender Bericht gemessen wurde. Im signierten Berichtspaket reist die Kopie als `references/rulesets.json`, gedeckt von Manifest und Signatur.

Ein Regelwerk mit Partnern teilen

Ein Leitsteuerungsanbieter kann ein Regelwerk mit den Fahrzeugherstellern teilen, die er als Partner freigegeben hat. Das Recht dazu gehört zum Vendor Programme: als eigener Plan, als Option zu Team und in Unlimited enthalten. Es ist dasselbe Recht, das Anforderungsdateien teilt. Ein Partner findet geteilte Regelwerke auf der Analyze-Seite unter *Rulesets shared with you*, nach Anbieter gruppiert.

Eine Anforderungsdatei kann ein Regelwerk beim Namen verlangen und über seinen SHA-256 festlegen. Der Hersteller wird dann an genau dieser Datei gemessen, mit den Parameterwerten, die die Anforderungsdatei setzt. Endet das Teilen, verschwindet das Regelwerk aus der Liste des Partners. Sonst ändert sich nichts: Ein bestehender Bericht behält seine Bewertung und seine eingefrorene Kopie.

Regeln schreiben, die echtem Verkehr standhalten

Vom Handbuch ausgehen, nicht vom Verkehr. Jede Prüfung ist ein Satz aus dem Dokument des Herstellers und `spec_ref` nennt Kapitel und Absatz. Eine Regel, die sich nicht belegen lässt, ist eine Meinung.

Die Beschreibung für den Leser des Befunds schreiben. Das ist ein Inbetriebnehmer um zwei Uhr nachts. Er muss wissen, was die Regel bedeutet, warum es sie gibt und was zu tun ist. `label` ist die Überschrift, `recommendation` der Ausweg.

Schweregrade wählen, wie der Hersteller sie wählen würde.

- `Critical` : Das Fahrzeug hält an oder lehnt den Auftrag ab.
- `Major` : Der Auftrag leidet oder das Verhalten liegt außerhalb der Spezifikation.
- `Minor` : Es funktioniert, verlässt aber die deklarierten Werte.
- `Warning` : laut zu sagen, aber kein Verstoß gegen den Standard.
- `Info` : sehenswert, aber keinen Streit wert.

Zuordnen, wo es geht. `addressee` macht aus *„es gibt einen Befund“* ein *„die Leitsteuerung hat das gesendet“*. Das ist der Unterschied zwischen einer Debatte und einer Behebung.

Eine Annahme sichtbar machen. Nennt das Handbuch keine Zahl, erklären Sie einen Parameter mit `assumed: true` . Der Bericht zeigt den Wert dann als Annahme, bis der Hersteller einen bestätigt.

Reguläre Ausdrücke verankern. `matches` sucht innerhalb des Texts, also lässt `"BIN-[0-9]{4}"` auch `XBIN-12345` durch. Schreiben Sie `^BIN-[0-9]{4}$` , wenn der ganze Wert gemeint ist.

Jede Regel an einem echten Mitschnitt testen. Eine Regel scheitert auf zwei Arten still. Sie **schlägt nie an**, weil ein Pfad vertippt ist, und der Bericht liest sich, als sei alles in Ordnung. Oder sie **schlägt immer an**, weil ein Prädikat zu scharf ist. Der Kunde ertrinkt dann in Befunden und schaltet das Regelwerk ab. Analysieren Sie einen Mitschnitt mit dem neuen Regelwerk und lesen Sie die Tabelle *Custom Ruleset Checks*. Bevor die Datei in eine Abnahme geht, sehen Sie zweimal hin: bei einer Prüfung mit *No matching messages*, wo Sie Verkehr erwartet haben, und bei einer mit Hunderten von Befunden.

Ein Regelwerk aus dem Integrationshandbuch eines Herstellers zu schreiben ist eine Leistung, die wir anbieten. Wir schreiben die Datei, lassen sie gegen einen Beispiel-Mitschnitt laufen, erklären beide Warnzeichen und übergeben sie. Sprechen Sie uns mit dem Dokument in der Hand an.

Grenzen

Grenze	Wert
Pläne	Project, Monitoring und Vendor Programme: 3 Regelwerke. Team und Unlimited: beliebig viele. Trial und Testbench: keine
Wer hochladen und löschen darf	Superuser und Administratoren
Prüfungen je Datei	200
Einträge je Katalog	200
Parameter je Datei	30
Dateigröße	2 MB
Befunde je Prüfung und Lauf	500, der Schnitt steht im Bericht
Offene Erwartungen je Regel vom Typ `response`	10.000 gleichzeitig, der Schnitt steht im Bericht
Kennung einer Prüfung	`^[A-Z][A-Z0-9_]{2,63}$`, Herstellerpräfix empfohlen
Sichtbarkeit	das ganze Enterprise; ein Regelwerk in einem Projekt nur dessen Mitgliedern

Fragen zu Custom Rulesets: **support@logiq-fleets.com** LogistiXpert · logistixpert.com