Reliable Context

RELIABLE CONTEXT / RESEARCH & PRACTICAL GUIDANCE

Data Contracts Explained: Producers, Consumers, Evidence

> Reliable Context is a kernel of truth: a system of record for organizations that fuse private and public information and must defend every value they act on. One canonical record per real thing, a cited source and an observation date behind every value, unknowns declared rather than left blank, and any past state of the record reproducible.

Published by Reliable Context · 2026-10-29 · data contracts · data contract example · open data contract standard · data contract vs schema · dbt model contracts

Original article: https://reliablecontext.com/articles/data-contracts-explained-producers-consumers-and-evidence



In this article

1. The failure data contracts address
2. Anatomy: schema, semantics, quality, SLAs, ownership
3. ODCS in practice
4. Enforcement in pipelines: dbt model contracts
5. Quality checks vs contracts: Great Expectations
6. Versioning and breaking changes
7. Contracts and provenance: who vouches
8. Adoption path
9. Sources

The failure data contracts address

A data contract is a machine-readable agreement about a dataset that one team produces and other teams or systems consume. The Open Data Contract Standard (ODCS) states it plainly: a data contract "defines the agreement between a data producer and consumers" (ODCS repository, README). The producer is the team that writes the data; the consumer is any report, model, pipeline or application that reads it.

The failure a contract addresses is an interface that nobody declared. A table becomes load-bearing for downstream work without anyone stating what it promises. The producer makes a locally reasonable change; the consumer discovers it later, often as a wrong number rather than an error. Such changes fall into three classes:

- **Structural**: a column is dropped, renamed or retyped. dbt's documentation notes that even a change from `boolean` to `integer` "could cause queries to fail in surprising ways" (dbt, contract configuration).
- **Semantic**: the structure is unchanged but the meaning moves. Consider a worked example at Halden Therapeutics AG (a fictional pharmaceutical company). Its quality-control (QC) laboratory changes `result_value` from milligrams per millilitre to micrograms per millilitre. The column is still numeric, so every type check passes, and every downstream trend chart is now off by a factor of 1,000.
- **Operational**: the data arrive late, stop arriving, or lose their owner.

A schema covers only the first class. A schema states representation: names, types, nullability, keys. A data contract adds what the schema cannot express: meaning, units, population, quality rules, freshness, ownership and a change policy. Aligning two schemas that use different names for the same thing is a separate task, schema alignment. A contract makes the producer's side of that mapping explicit.

Anatomy: schema, semantics, quality, SLAs, ownership

The ODCS repository lists the sections of a contract as fundamentals, schema, references, data quality, support and communication channels, pricing, team, roles, service-level agreement, infrastructure and servers, authoritative definitions, tags, and custom properties (ODCS repository, README). Version 3.2.0 added an

optional `context` block with guidance for AI agents and BI tools (ODCS changelog). The table maps consumer questions to ODCS v3.2.0 keys.

Consumer question	ODCS v3.2.0 keys	Typically checked by
Which contract, which revision, which lifecycle stage?	`apiVersion`, `kind`, `id`, `version`, `status` (Fundamentals)	Contract registry, CI
What is the shape?	`schema[]` objects and `properties[]` with `logicalType`, `physicalType`, `required`, `primaryKey`, `unique`, `enum` (Schema)	Build-time contract, DDL
What does a value mean?	`description` (`purpose`, `limitations`, `usage`), `businessName`, `authoritativeDefinitions`	Human review, linked definitions
What content is acceptable?	`quality[]` of type `library`, `sql`, `text` or `custom` (Data Quality)	Great Expectations, dbt tests, SQL
How fresh, how long kept?	`slaProperties[]` such as `latency`, `frequency`, `retention`, `timeOfAvailability` (SLA)	Monitoring
Who answers for it?	`team` with `members[]`, `support[]` (Team, Support)	Organisational process
Who may read it, where is it?	`roles[]` (Roles), `servers[]` (Servers)	IAM, platform
What does it cost?	`price` with `priceAmount`, `priceCurrency`, `priceUnit` (Pricing)	Billing

Two details matter. First, the companion JSON Schema for v3.2.0 makes only four top-level keys mandatory: `version`, `apiVersion`, `kind` and `id` (ODCS JSON Schema v3.2.0). Second, ODCS has no unit key on schema properties; in that JSON Schema, `unit` appears only on quality rules and SLA properties. A unit must therefore be carried by a description, a separate unit column with a quality rule, or a custom property. The example below uses the second option, which a machine can check.

ODCS in practice

ODCS is published by Bitol under the Apache 2.0 licence, and its current version is v3.2.0 (ODCS repository, README), released on 8 September 2026 (GitHub release v3.2.0). In ODCS terms, an *object* is a structure such as a table or document, a *property* is an attribute such as a column, and an *element* is either (Schema).

The contract below is for the fictional Halden Therapeutics AG QC results table. It validates without errors against the v3.2.0 JSON Schema (checked on 2026-10-28). The ODCS documentation notes that the JSON Schema is a companion and that the standard prevails in a conflict (ODCS, Definition).

```
# Fictional example: Halden Therapeutics AG is not a real company.
apiVersion: v3.2.0
kind: DataContract
id: 7f3c2a10-5b8e-4d6a-9c41-2e0f8b7d9a55
name: qc_assay_results
version: 2.0.0
status: active
```

```yaml
domain: quality-control
description:
  purpose: Released QC assay results per drug-product batch, for release dashboards and stability trending.
  limitations: Excludes out-of-specification results still under laboratory investigation.
  usage: Join to batch master data on batch_id; filter on measured_at for time ranges.
schema:
  - id: qc_assay_results_obj
    name: qc_assay_results
    physicalName: qc_assay_results_v2
    physicalType: table
    logicalType: object
    dataGranularityDescription: One row per batch, assay and test occasion.
    properties:
      - id: batch_id_prop
        name: batch_id
        logicalType: string
        physicalType: varchar(12)
        required: true
        criticalDataElement: true
        logicalTypeOptions:
          pattern: '^HT-[0-9]{4}-[0-9]{5}$'
      - id: assay_code_prop
        name: assay_code
        logicalType: string
        required: true
        enum:
          - value: ASSAY_HPLC
            description: Content of active substance by HPLC.
          - value: WATER_KF
            description: Water content by Karl Fischer titration.
      - id: result_value_prop
        name: result_value
        businessName: Assay result
        logicalType: number
        physicalType: numeric(12,4)
        required: true
        description: Reported result, in the unit given by result_unit.
      - id: result_unit_prop
        name: result_unit
        logicalType: string
        required: true
        quality:
          - id: result_unit_allowed
            metric: invalidValues
            arguments:
              validValues: ['mg/mL', '% w/w']
            mustBe: 0
            dimension: conformity
            severity: error
      - id: measured_at_prop
        name: measured_at
        logicalType: timestamp
```

```
        required: true
        logicalTypeOptions:
          timezone: true
    quality:
      - id: no_duplicate_tests
        metric: duplicateValues
        arguments:
          properties: [batch_id, assay_code, measured_at]
        mustBe: 0
        dimension: uniqueness
slaProperties:
  - id: latency_1d
    property: latency
    value: 1
    unit: d
    element: qc_assay_results.measured_at
    driver: regulatory
team:
  name: QC Data Products
  members:
    - username: qc-data-owner
      role: Owner
support:
  - channel: '#qc-data-contracts'
    tool: slack
servers:
  - server: qc-prod
    type: postgres
    environment: prod
    host: qc-db.halden.example
    port: 5432
    database: qc
    schema: released
```

`apiVersion` names the version of ODCS, while `version` is the version of this contract (Fundamentals). Each element carries an `id`, which ODCS recommends for "stable, refactor-safe references" (Schema). The `enum` block is new in v3.2.0 (ODCS changelog). The library metrics `invalidValues` and `duplicateValues` come from the predefined ODCS metric set, and `dimension` takes values such as `conformity` and `uniqueness` (Data Quality). In the SLA, ODCS lists `latency` as "preferred to freshness". The `driver` field records whether a commitment is regulatory, analytical or operational (SLA).

This contract leaves semantics deliberately thin. Whether `ASSAY_HPLC` needs a glossary entry, a taxonomy or an ontology depends on the decision the consumer makes. That question is the subject of the next-step article, When a Data Contract Needs an Ontology: ODCS 3.2 Test.

Enforcement in pipelines: dbt model contracts

A contract document enforces nothing; enforcement happens where data are built. In dbt, a *model contract* is a set of guarantees about a model's shape: "dbt will verify that your model's transformation will produce a dataset matching up with its contract, or it will fail to build" (dbt, Model contracts). The Halden table, expressed as a dbt model contract:

```
models:
  - name: qc_assay_results
    config:
      materialized: table
      contract:
        enforced: true
    columns:
      - name: batch_id
        data_type: varchar(12)
        constraints:
          - type: not_null
      - name: assay_code
        data_type: varchar(20)
        constraints:
          - type: not_null
      - name: result_value
        data_type: numeric(12,4)
      - name: result_unit
        data_type: varchar(10)
      - name: measured_at
        data_type: timestamp
```

When `enforced: true` is set, the contract must list every column's `name` and `data_type`. dbt then does two things. First, it runs a "preflight" check that the query returns columns with matching names and data types, regardless of column order. Second, it puts column names, types and constraints into the DDL it submits, so the platform enforces them during the build (dbt, Model contracts). A mismatch fails before the table is materialized (dbt, contract configuration).

Constraint enforcement depends on the platform; some constraints are definable but recorded only as metadata (dbt, Model contracts):

Constraint	Snowflake	BigQuery	Redshift	Postgres	Spark
`not_null`	enforced	enforced	enforced	enforced	definable, not enforced
`primary_key`	metadata only	metadata only	metadata only	enforced	definable, not enforced
`foreign_key`	metadata only	metadata only	metadata only	enforced	definable, not enforced
`unique`	metadata only	not definable	metadata only	enforced	definable, not enforced
`check`	not definable	not definable	not definable	enforced	definable, not enforced

What a dbt contract does not guarantee is equally specific:

- **Size, precision and scale.** dbt does not compare these, so `varchar(12)` versus `varchar(13)` passes (dbt, contract configuration).
- **Values.** No contract checks that `result_unit` holds `mg/mL`, that a pattern matches, or that a range holds. dbt directs that work to data tests, which validate content after the build (dbt, Model contracts).
- **Coverage.** Views support names and types but not constraints. Incremental models need `on_schema_change` set to `append_new_columns` or `fail`. Python models, ephemeral models, sources, seeds and snapshots are not supported (dbt, Model contracts).
- **Meaning, freshness and ownership.** These are outside the feature. dbt says the API analogy is deliberate: the response structure is the contract, while quality and uptime "are not part of the contract per se" (dbt, Model contracts).

Quality checks vs contracts: Great Expectations

The contract declares; a quality engine evaluates. Great Expectations (GX) Core is a Python library for data validation workflows. Its vocabulary maps onto the contract (GX Core overview):

- An **Expectation** is "a verifiable assertion about data". An **Expectation Suite** is a collection of them.
- A **Data Source** describes how to connect to a data store. A **Data Asset** is a collection of records in it, such as a table. A **Batch Definition** organises those records into **Batches** for validation.
- A **Validation Definition** binds a Batch Definition to an Expectation Suite. A **Checkpoint**, the primary means of production validation, runs Validation Definitions and passes **Validation Results** to **Actions** such as alerts or updates to **Data Docs**, the human-readable record.

The `result_unit` rule in the Halden contract corresponds to `ExpectColumnValuesToBeInSet` (GX Expectation Gallery) with `value_set` set to the two permitted units (GX source, expect_column_values_to_be_in_set.py). GX adds two parameters a contract author must decide on. `mostly` sets the fraction of rows that must pass. `severity` takes `critical`, `warning` or `info` (GX, Create an Expectation). Record both in the contract, or the tolerance lives only in code. ODCS can also carry GX configuration directly: a `custom` quality rule with `engine: greatExpectations` passes an `implementation` block to the tool unchanged (Data Quality).

The division of labour: the contract is the reviewed, versioned statement of what must hold; dbt's build-time contract rejects structural violations; GX or dbt data tests evaluate content; each run leaves a Validation Result recording what was checked, on which batch, with what outcome.

Versioning and breaking changes

ODCS records the contract's own revision in `version` and its lifecycle in `status`, with example values `proposed`, `draft`, `active`, `deprecated` and `retired` (Fundamentals). Since v3.2.0, any object or property can set `deprecated: true`; deprecated elements "remain documented and validated for backward compatibility"

(Schema). The ODCS sections reviewed here do not define which changes are breaking. Teams often borrow Semantic Versioning, where a major version "MUST be incremented if any backward incompatible changes are introduced" (Semantic Versioning 2.0.0).

dbt is more specific for the structural layer. When it compares a project with its previous state, it treats as breaking: removing a column, changing a column's `data_type`, removing or modifying a constraint, and removing a contracted model. Versioned models raise an error and unversioned models a warning (dbt, Model contracts). Intentional breaking changes call for a new model version. Adding a column or fixing a bug in a calculation does not (dbt, Model versions). Model versions let consumers pin a version. A `deprecation_date` bounds the migration window (dbt, Model versions).

The gap is semantic change, which no structural diff detects. Applied to the Halden contract:

Change	Detected by a dbt contract diff?	Recommended contract version bump
Add nullable column `analyst_id`	No (non-breaking in dbt)	Minor
Remove column `result_unit`	Yes, breaking	Major
`result_value` from `numeric` to `varchar`	Yes, breaking	Major
`varchar(12)` to `varchar(16)` on `batch_id`	No (size not compared)	Minor
`result_value` reported in µg/mL instead of mg/mL	No	Major
New `assay_code` value `DISSOLUTION`	No	Major for consumers with exhaustive logic; otherwise minor
Latency commitment from 1 day to 3 days	No	Major

The "Recommended" column is this article's judgement, not an ODCS or dbt rule: the producer, not the tool, must classify semantic and operational changes, and the contract change review is where that classification is recorded.

Contracts and provenance: who vouches

A contract is a promise made by a named owner. The `team` block records members with `role`, `dateIn`, `dateOut` and `replacedByUsername` (Team), so the promise has a history as well as an author. A promise is not evidence. Three kinds of evidence sit beside it:

1. **Build evidence**: the dbt run that passed or failed the preflight check.
2. **Validation evidence**: GX Validation Results and Data Docs for each batch.
3. **Lineage evidence**: which job run read which inputs and wrote which outputs. OpenLineage, an open specification, models this with dataset, job and run entities that are extended through facets (OpenLineage documentation).

Passing every check shows that the asserted properties held for the batch that was checked. It does not show that a value is true. A `result_value` of 98.7 mg/mL can satisfy every rule in the contract and still differ from the instrument record it was copied from. Establishing that requires source-level provenance for the value itself, which is the line between describing datasets and holding verified values. Documentation frameworks such as Datasheets for Datasets add motivation, composition and collection process (Gebru et al., Datasheets for Datasets). For data published as RDF graphs, the W3C Shapes Constraint Language (SHACL) plays the validation role that dbt and GX play for tables (W3C, SHACL).

Adoption path

1. **Pick one interface with a known consumer.** dbt recommends contracts for "public" models relied on downstream. It also warns that governance features add maintenance if adopted too early (dbt, Model contracts).
2. **Write fundamentals, schema and owner.** Give every property `logicalType` and `required`; name an owner in `team` and a `support` channel.
3. **Enforce structure at build time.** Mirror the schema in a dbt model contract, materialized as a table, and know which constraints your platform actually enforces.
4. **Add a small number of executable quality rules.** Start with rules that would have caught recent incidents; declare them in `quality`, execute them in GX or dbt tests, and keep the Validation Results.
5. **Declare one SLA.** `latency` on the event timestamp is usually the most informative.
6. **Write the change policy.** State the versioning scheme, the deprecation window, and the rule that unit, definition and population changes are major.
7. **Then address meaning.** Once consumers dispute what a field denotes, work through the acceptance test in When a Data Contract Needs an Ontology: ODCS 3.2 Test before choosing a glossary link, a taxonomy or an ontology.

Reliable Context is a system of record built around source-backed values, which addresses the evidence layer that sits beside a data contract.

Sources

1. Open Data Contract Standard, repository README (v3.2.0). Bitol. https://github.com/bitol-io/open-data-contract-standard
2. Open Data Contract Standard, release v3.2.0. Bitol, GitHub, 8 September 2026. https://github.com/bitol-io/open-data-contract-standard/releases/tag/v3.2.0
3. Definition: Open Data Contract Standard (ODCS). Bitol. https://bitol-io.github.io/open-data-contract-standard/latest/
4. ODCS, Fundamentals. Bitol. https://bitol-io.github.io/open-data-contract-standard/latest/fundamentals/
5. ODCS, Schema. Bitol. https://bitol-io.github.io/open-data-contract-standard/latest/schema/
6. ODCS, Data Quality. Bitol. https://bitol-io.github.io/open-data-contract-standard/latest/data-quality/
7. ODCS, Service-Level Agreement. Bitol. https://bitol-io.github.io/open-data-contract-standard/latest/service-level-agreement/

8. ODCS, Team. Bitol. https://bitol-io.github.io/open-data-contract-standard/latest/team/
9. ODCS, Support & Communication Channels. Bitol. https://bitol-io.github.io/open-data-contract-standard/latest/support-communication-channels/
10. ODCS, Roles. Bitol. https://bitol-io.github.io/open-data-contract-standard/latest/roles/
11. ODCS, Infrastructure & Servers. Bitol. https://bitol-io.github.io/open-data-contract-standard/latest/infrastructure-servers/
12. ODCS, Pricing. Bitol. https://bitol-io.github.io/open-data-contract-standard/latest/pricing/
13. ODCS, Changelog. Bitol. https://bitol-io.github.io/open-data-contract-standard/latest/changelog/
14. ODCS JSON Schema v3.2.0. Bitol, GitHub. https://github.com/bitol-io/open-data-contract-standard/blob/main/schema/odcs-json-schema-v3.2.0.json
15. Model contracts. dbt Labs, dbt Developer Hub. https://docs.getdbt.com/docs/mesh/govern/model-contracts
16. contract (resource configuration reference). dbt Labs, dbt Developer Hub. https://docs.getdbt.com/reference/resource-configs/contract
17. Model versions. dbt Labs, dbt Developer Hub. https://docs.getdbt.com/docs/mesh/govern/model-versions
18. GX Core overview (GX Core 1.24.1). Great Expectations. https://docs.greatexpectations.io/docs/core/introduction/gx_overview/
19. Create an Expectation (GX Core 1.24.1). Great Expectations. https://docs.greatexpectations.io/docs/core/define_expectations/create_an_expectation
20. ExpectColumnValuesToBeInSet, Expectation Gallery. Great Expectations. https://greatexpectations.io/expectations/expect_column_values_to_be_in_set
21. expect_column_values_to_be_in_set.py, great_expectations source. Great Expectations, GitHub. https://github.com/great-expectations/great_expectations/blob/develop/great_expectations/expectations/core/expect_column_values_to_be_in_set.py
22. Semantic Versioning 2.0.0. T. Preston-Werner. https://semver.org/spec/v2.0.0.html
23. About OpenLineage (v1.54.0). OpenLineage project. https://openlineage.io/docs/
24. Gebru, T. et al. Datasheets for Datasets. arXiv:1803.09010; published in Communications of the ACM, December 2021. https://arxiv.org/abs/1803.09010
25. Shapes Constraint Language (SHACL). W3C Recommendation, 20 July 2017. https://www.w3.org/TR/shacl/

Source links

Links cited in this article, in order of first appearance. Full addresses are provided for readers using extracted text.

1. https://github.com/bitol-io/open-data-contract-standard
2. https://docs.getdbt.com/reference/resource-configs/contract
3. https://bitol-io.github.io/open-data-contract-standard/latest/changelog
4. https://bitol-io.github.io/open-data-contract-standard/latest/fundamentals
5. https://bitol-io.github.io/open-data-contract-standard/latest/schema

6. https://bitol-io.github.io/open-data-contract-standard/latest/data-quality
7. https://bitol-io.github.io/open-data-contract-standard/latest/service-level-agreement
8. https://bitol-io.github.io/open-data-contract-standard/latest/team
9. https://bitol-io.github.io/open-data-contract-standard/latest/support-communication-channels
10. https://bitol-io.github.io/open-data-contract-standard/latest/roles
11. https://bitol-io.github.io/open-data-contract-standard/latest/infrastructure-servers
12. https://bitol-io.github.io/open-data-contract-standard/latest/pricing
13. https://github.com/bitol-io/open-data-contract-standard/blob/main/schema/odcs-json-schema-v3.2.0.json
14. https://github.com/bitol-io/open-data-contract-standard/releases/tag/v3.2.0
15. https://bitol-io.github.io/open-data-contract-standard/latest
16. https://docs.getdbt.com/docs/mesh/govern/model-contracts
17. https://docs.greatexpectations.io/docs/core/introduction/gx_overview
18. https://greatexpectations.io/expectations/expect_column_values_to_be_in_set
19. https://github.com/great-expectations/great_expectations/blob/develop/great_expectations/expectations/core/expect_column_values_to_be_in_set.py
20. https://docs.greatexpectations.io/docs/core/define_expectations/create_an_expectation
21. https://semver.org/spec/v2.0.0.html
22. https://docs.getdbt.com/docs/mesh/govern/model-versions
23. https://openlineage.io/docs
24. https://arxiv.org/abs/1803.09010
25. https://www.w3.org/TR/shacl

THE PUBLISHER

About Reliable Context

Reliable Context is a system of record for organizations that fuse private and public information and must defend every value they act on.

The guarantees

One canonical record per real thing: duplicates resolve by rule, ambiguity is shown rather than hidden, and related organizations relate without ever being merged. A verbatim, cited source and an observation date sit behind every value. Unknowns are declared as not found, blocked or not applicable with a reason, never left as an empty cell. Any past state of the record is reproducible, together with what changed since.

Where it runs

It runs inside the customer's perimeter, one database per organization. Permitted sources, model routing, reader clearance and retention are set as the customer's own policies, and every value is exportable with its chain of custody for an auditor or regulator.

What it is not

It is not a vector database or a RAG framework, not a data catalog, MDM or data-quality suite, not a CRM or data-enrichment vendor, and not a chatbot or AI assistant. Where a reader needs one of those, the honest answer is to say so and point them there.

Visit

See Reliable Context.

Reliable Context is in production use in two sectors; more than six thousand of its identity decisions were made by rule and none has been reversed. Articles published here explain general practice, cite their primary and academic sources, and name the jurisdiction of any regulatory rule they describe. They are not legal, regulatory or compliance advice for a specific organization, and requirements change.

Website: https://reliablecontext.com

This article is educational. Verify consequential medical, legal, financial, regulatory and technical decisions with appropriate professionals and the cited primary sources. Company services are described separately from the article's research findings.