Define what crosses the boundary.

An integration contract covers meaning, ownership and failure handling as well as the shape of a message.

Customer update contract

Contract field	Example rule
customer_id	Stable identifier shared by both systems
field_authority	CRM owns relationship fields, ERP owns credit status
event_version	The schema version used to interpret the payload
retry_policy	Repeat transport safely without repeating the business action

Specify absent and cleared values

An omitted field, an empty value and an explicit deletion can mean different things. Document each case so an update cannot accidentally erase valid data.

Evolve the contract deliberately

Test existing consumers before releasing a change. Use a migration window when meaning or required fields change.

Agree what each record means

An interface contract must explain meaning as well as format. Agree which system owns each field, which identifiers remain stable and when a change becomes authoritative. For an order integration, distinguish accepted, paid, dispatched and cancelled states. Treating all four as a generic update makes reconciliation and support unnecessarily difficult.

Document units, currency, time zones and the interpretation of missing values. Decide whether an omitted field means unchanged, unknown or clear the existing value. Include examples covering those distinctions. A payload can be technically valid and still produce an incorrect business change if the two systems interpret it differently. Record the business owner who can resolve an ambiguity in the contract.

Define uncertainty and repeated delivery

Specify authentication, permitted operations and request limits. Explain how a caller identifies a repeated request and how the receiver handles a retry after a timeout. If events can arrive more than once or out of order, describe the consumer's responsibilities. Include the lifetime of deduplication records and the conditions under which replay is safe.

Different failures need different responses. A validation error needs correction, a temporary dependency failure may justify a delayed retry, and an uncertain write outcome may require reconciliation first. Agree which failures are shown to users, which enter an operations queue and which trigger an alert. Do not leave each consuming application to invent its own interpretation.

Maintain the agreement as systems change

Assess changes by their effect on consumers. Adding a field may be harmless for one application and break another that rejects unknown properties. Test representative consumers and agree a transition period where necessary. Record who owns the interface, how changes are announced and where consumers report problems.

Before release, exercise a normal request, a duplicate, a malformed record and an interrupted response. Confirm that the resulting business state matches the contract in each case. Preserve these examples as reusable tests. During an incident, the contract should help teams identify affected records and responsibility for the next action, without first debating how the interface was supposed to work.

Who approves a field mapping?

The owner of the business data, with engineering confirming the technical representation and verification approach.

Read the current resource online

https://cobnex-review.rgcsekaraa.workers.dev/resources/integration-contracts/