Chatwoot mutation approval and idempotency

This reference gives one governance model for all 153 Chatwoot capabilities at the reviewed revision. Use it to decide when approval is required, how to bind an idempotency key, and what to do after an uncertain result.

SECTION	Connectors
TYPE	Connector
PROTOCOL	AIP 1.0
SOURCE	docs/connectors/chatwoot/reference/mutation-approval-and-idempotency.md

Approval authorizes one reviewed intent. Idempotency correlates attempts for that intent. Neither control proves that Chatwoot applied the intended state.

Capability population

Population	Count	Approval	Published retry
Catalogue GET reads	62	Not required	Safe
Catalogue POST, PATCH, and DELETE operations	88	Required	Unsafe
`cap:chatwoot:message:create`	1	Required	Unsafe
`cap:chatwoot:conversation:status`	1	Required	Safe
`cap:chatwoot:conversation:handoff`	1	Required	Unsafe
All capabilities	**153**	**91 require approval**	**63 safe, 90 unsafe**

The catalogue classifies every non-GET provider method as a mutation. This includes read-like names backed by POST, such as `contact.filter`, and operational commands backed by POST.

Risk and approval are related but not identical. Every catalogue mutation requires approval whether its published risk is medium or high. The three connector composites also require approval.

Shared mutation contract

All 91 mutations publish:

Contract field	Reviewed value
Approval	Required
Approver selector	Tenant policy
Approval validity	900,000 ms
Approval evidence	Reason, input snapshot, policy decision
Idempotency	Required
Key scope	External account

Contract field	Reviewed value
Collision behavior	Revalidate input hash
Published key lifetime	86,400,000 ms
Data	Restricted, PII present, redaction required
Completion	Synchronous
Compensation	Rollback not supported

Catalogue mutations also publish policy version `chatwoot-production-mutation-v1`. The three composite approval policies do not set that connector-local version field. Runtime and tenant policy still own the admitted approval decision.

Decide before creating an Action

For each intended provider change, verify these conditions in order:

1. Confirm the selected capability is admitted for this tenant, connector

version, instance, and external account.

2. Prefer a GET capability when it satisfies the task without changing

provider state.

3. Confirm every path identity is authoritative and tied to the configured

account.

4. Match the provider-owned body to the pinned Chatwoot route.
5. Identify every customer-visible, destructive, relationship, automation, or

disclosure effect that can occur.

6. Select the final read that can establish the intended state.
7. Select the recovery decision for a lost provider response.

Stop before Action creation when any answer is unknown. Approval should not be used to compensate for an unpinned body or ambiguous customer identity.

Bind approval to one immutable intent

The approval record needs enough evidence to distinguish this mutation from every nearby alternative:

Evidence	Bind to
Actor and tenant	Authenticated AIP identity
Route	Connector type, version, instance, and external account
Capability	Exact ID, including dots or colons
Input snapshot	Canonical path inputs, query, body, or multipart identity
Reason	Human-readable business purpose
Policy decision	Policy version, hash, constraints, and decision identity
Effect review	Visibility, destination, deletion, merge direction, or automation consequence
Verification plan	Authoritative read and fields checked after settlement

An approval is not a reusable permission for edited input. Changing message text, `private` visibility, status, assignee, record identity, relationship direction, filter, import, export, merge participants, or provider body creates a

different intent.

The published approval lifetime is 15 minutes. Expiry does not establish that the provider call did or did not run. Consult Action state and provider evidence before requesting a replacement decision.

Design the idempotency key

One mutation intent receives one stable key. The published scope is the configured external account, so the application should include enough business context to avoid collision inside that account.

Readable examples:

```
TEXT
chatwoot:conversation:314:status:pending:v1
chatwoot:company:88:contact:205:detach:v1
chatwoot:contact-merge:source-205:destination-401:v1
```

Do not place customer names, email addresses, message text, tokens, or other PII in the key. Store the canonical input hash next to the key instead.

The catalogue connector accepts a key only when it:

- is present for a mutation;
- is not blank after trimming;
- is at most 512 bytes;
- contains no control characters.

The protocol collision behavior is `revalidate_input_hash`. Reusing a key with different canonical input is a collision, not a request to update the original intent.

The published 24-hour lifetime bounds retained idempotency behavior. It is not permission to recycle a business key after one day. Use a new versioned key for a genuinely new intent.

Catalogue enforcement and forwarding

For the 150 dot-named catalogue capabilities, invocation performs these steps:

1. classify the operation by its frozen provider method;
2. require and validate a key for non-GET methods;
3. build the account-scoped provider URL;
4. add `X-AIP-Action-ID` with the Action ID;
5. add `Idempotency-Key` when a key is present;
6. send the bounded query, JSON body, or multipart body;
7. return provider status, body, and request ID when exposed.

A read may carry an optional key, which the connector also forwards. All 62 GET reads advertise safe retry.

Header forwarding establishes correlation at the connector boundary. The reviewed code does not prove that every Chatwoot route stores, interprets, or deduplicates `Idempotency-Key`.

Composite enforcement boundary

The three colon-named composites publish required external-account idempotency, but their direct connector branches do not call the catalogue key validator.

They also do not send `Idempotency-Key` or `X-AIP-Action-ID` headers to Chatwoot. Message and handoff instead place the Action ID inside connector content attributes when they create a message.

Composite	Provider effect	Direct key enforcement	Provider key header	Retry
`message:create`	Create public reply or private note	No	No	Unsafe
`conversation:status`	Set open, resolved, or pending	No	No	Safe
`conversation:handoff`	Assign, then optionally create private note	No	No	Unsafe

Use the governed AIP path so capability contract, identity, policy, approval, and idempotency admission occur before connector invocation. Direct connector invocation is not a substitute for that gate.

Interpret safe retry narrowly

Published safe retry means the contract permits a correlated repeat under its stated scope. It does not mean repeat without reading current state.

Operation class	Before a repeat
GET catalogue read	Reuse or create a read Action according to caller policy; expect a new provider snapshot
`conversation:status`	Read the conversation; accept matching state before considering the same intent
Any other catalogue mutation	Reconcile provider state and retained request identity; do not replay blindly
`message:create`	Read messages and correlate AIP content attributes where exposed
`conversation:handoff`	Read assignment and messages separately because the composite can partially complete

All connector failures currently set `retryable: false`. This failure field is conservative and does not erase the capability-level safe-retry declaration for reads or status. Use both the contract and authoritative current state.

Keep Action, key, and input together

For one mutation, retain:

```
JSON
{
  "action_id": "act_chatwoot_example_001",
  "capability_id": "cap:chatwoot:example.operation",
  "idempotency_key_hash": "sha256:...",
  "canonical_input_hash": "sha256:...",
  "approval_id": "appr_...",
  "policy_hash": "sha256:...",
  "connector_instance": "instance:...",
  "external_account": "account:...",
  "provider_request_id": null
}
```

This is an evidence shape, not an AIP request schema. Keep the raw key and restricted input in an appropriately protected store. Do not log credentials or full customer data for convenience.

When a response is lost, query durable Action state with the original Action ID. Do not create a replacement Action merely to obtain a new response.

Interpret completion and failure

A catalogue completion proves that the provider returned a successful HTTP status within the active response bound. A composite completion proves its implemented provider calls returned success and the connector constructed its result.

Neither result proves:

- customer-channel delivery;
- absence of later provider processing;
- atomic handoff;
- provider-side idempotency retention;
- intended state without a final read.

Provider `429` and server responses, transport failures, cancellation, and oversized invocation responses can mark outcome uncertain. The provider may have accepted a mutation before the connector lost or rejected its response.

Reconcile by effect

Mutation effect	Authoritative evidence	Safe decision
Create	Search or list using stable provider fields	Accept one matching object or escalate; do not create another blindly
Update or status	Read the exact object	Accept matching state or create a new corrective intent
Send message or note	List messages and correlate Action attributes	Accept the existing message or escalate; do not resend blindly
Attach or detach relationship	List the relationship from its owning object	Accept current state or create a separately approved correction
Merge	Read destination, source when addressable, and material relationships	Accept reconciled state or escalate; never submit another merge blindly
Import or export	Provider job, artifact, and sampled record evidence	Determine whether work or disclosure already exists
Delete	Object absence plus audit and dependent state	Correlate the original Action; absence alone does not identify the actor
Handoff	Assignment first, then private-note evidence	Resolve partial effects independently

No Chatwoot mutation publishes an AIP rollback capability. A corrective provider operation is a new Action with a new approval and a key for that new intent.

Verification checklist

Before declaring a mutation settled, require:

- exact admitted capability, connector version, instance, and account;
- one canonical input and input hash;
- one Action ID and one stable idempotency key;
- one approval matching the exact input and effect;
- no changed input under the original key;
- a terminal AIP state correlated to the Action;
- provider status and request identity where exposed;
- an authoritative final read for the intended effect;

- explicit resolution of any uncertain or partial outcome;
- restricted-data redaction and retention.

These controls validate use of the reviewed connector contract. They do not establish provider deduplication, production readiness, or live qualification.

Related documentation

- Chatwoot capability index
- AIP composite operations
- Actions and sessions
- Approvals and policy
- Error reference