Manage a Chatwoot conversation lifecycle

Use this guide to read one existing Chatwoot conversation, send a public reply or private note, set its status, optionally hand it off, and verify final provider state. An optional branch covers creation when the pinned provider body is alr

SECTION	Connectors
TYPE	Connector
PROTOCOL	AIP 1.0
SOURCE	docs/connectors/chatwoot/guides/conversation-and-message-lifecycle.md

The procedure preserves one Action identity and one idempotency key for each mutation. It never treats a lost response as permission to create a replacement Action.

When to use this guide

Use this procedure when an application owns one bounded conversation task and needs an auditable path through Chatwoot. It is suitable for a human-approved support reply, status change, or handoff.

Use the exhaustive conversation and message reference when choosing another provider operation. Use the composite reference when implementing the strict inputs themselves.

This guide does not configure the connector, create provider credentials, ingest webhooks, prove customer delivery, or qualify a Chatwoot deployment.

Prerequisites

Before starting, obtain:

- an authenticated AIP caller bound to the intended tenant;
- an admitted Chatwoot connector route for one configured account;
- a conversation ID from that same account, unless creating one;
- authority to request and approve each intended mutation;
- a private evidence location for Action, approval, and provider identifiers;
- the exact pinned Chatwoot body schema if using `conversation.create`.

Apply two application checks required by the pinned source:

- always send `private` explicitly to `cap:chatwoot:message:create`;
- require at least one of `assignee_id` or `note` for

`cap:chatwoot:conversation:handoff`.

Do not use direct connector invocation as an idempotency gate. The composite branches publish a required key contract but do not validate or forward that key themselves. Use the governed AIP path.

Plan the lifecycle

Step	Capability	Risk	Approval	Retry
Read conversation	`cap:chatwoot:conversation.get`	Low	No	Safe
Read messages	`cap:chatwoot:message.list`	Low	No	Safe
Create conversation, optional	`cap:chatwoot:conversation.create`	Medium	Required	Unsafe
Send reply or private note	`cap:chatwoot:message:create`	Medium	Required	Unsafe
Set status	`cap:chatwoot:conversation:status`	Medium	Required	Safe
Handoff, optional	`cap:chatwoot:conversation:handoff`	High	Required	Unsafe

The two reads are dot-named catalogue capabilities. The message, status, and handoff operations are colon-named connector composites. Do not normalize the IDs.

1. Pin discovery and identity

Read tenant capability discovery and require the exact IDs planned above. Record:

- tenant and authenticated actor;
- admitted connector type and immutable version;
- connector instance and configured external account;
- capability IDs and catalogue revision;
- credential revision without credential bytes;
- policy version used for approval.

Stop if the intended capability is absent. An operations file can filter dot-named catalogue operations, while the connector-generated manifest always starts with the three composites. Tenant admission and authorization remain separate controls.

Do not put `account_id` in Action input. The connector host inserts its configured account into every provider route.

2. Read authoritative before-state

Read the conversation:

```
SH
CONVERSATION_ID=314

aipctl action call \
  "$AIP_URL" \
  cap:chatwoot:conversation.get \
  --action-id act_chatwoot_lifecycle_read_001 \
  --input "{\"conversation_id\":\"$CONVERSATION_ID\"}"
```

Then read its messages:

```
SH
aipctl action call \
  "$AIP_URL" \
  cap:chatwoot:message.list \
  --action-id act_chatwoot_lifecycle_messages_001 \
  --input "{\"conversation_id\":\"$CONVERSATION_ID\"}"
```

Require completed results tied to the submitted Action IDs. Preserve provider HTTP status, response body, provider request ID when present, and observation time.

Confirm that the conversation belongs to the intended account and that current status, assignee, recent messages, and other fields needed by policy match the planned change. Provider result fields are not frozen by the generic output schema.

3. Create a conversation when needed

Skip this step for an existing conversation.

`cap:chatwoot:conversation.create` uses:

```
TEXT
POST /api/v1/accounts/{configured_account_id}/conversations
```

Its Action input has this source-owned envelope:

```
TEXT
{
  "body": <exact object for the pinned Chatwoot route>
}
```

The connector does not validate provider-specific creation fields. Build the body from the pinned provider schema, then retain its canonical form.

Before dispatch:

1. assign one Action ID;
2. assign one external-account-scoped idempotency key;
3. bind approval to actor, tenant, capability, exact body, and policy hash;
4. keep the same Action, key, and body through settlement.

The provider call is unsafe to retry and has no rollback. If its response is lost, search or list provider state before another create. Extract the new conversation ID only from a completed result or authoritative read.

After creation, repeat the two before-state reads with that conversation ID.

4. Send one reply or private note

Choose visibility deliberately:

Intent	`private`
Customer-visible reply	`false`
Internal note	`true`

Prepare one exact input:

```
JSON
{
  "conversation_id": "314",
  "content": "Your request is now with the operations team.",
  "private": false
}
```

The pinned runtime can reject an omitted `private` even though the published schema carries a default. Keep the Boolean explicit.

Submit with one Action and key:

```
SH
MESSAGE_ACTION=act_chatwoot_lifecycle_message_001
```

```
MESSAGE_KEY=chatwoot-lifecycle-message-314-v1

aipctl action call \
  "$AIP_URL" \
  cap:chatwoot:message:create \
  --action-id "$MESSAGE_ACTION" \
  --idempotency-key "$MESSAGE_KEY" \
  --input "{\"conversation_id\":\"$CONVERSATION_ID\",\"content\":\"Your request is now with the
operations team.\",\"private\":false}"
```

Expected governed flow: the Action can enter `pending_approval` before provider dispatch. Approve only the exact text and visibility. A changed character or Boolean requires a new decision.

After provider HTTP success, the composite constructs:

```
JSON
{
  "conversation_id": "314",
  "sent": true
}
```

This does not prove customer-channel delivery. The provider message contains AIP connector, loop-prevention, and Action attributes for later correlation.

5. Set the intended status

Choose exactly one status:

```
TEXT
open | resolved | pending
```

Prepare and submit:

```
SH
STATUS_ACTION=act_chatwoot_lifecycle_status_001
STATUS_KEY=chatwoot-lifecycle-status-314-pending-v1

aipctl action call \
  "$AIP_URL" \
  cap:chatwoot:conversation:status \
  --action-id "$STATUS_ACTION" \
  --idempotency-key "$STATUS_KEY" \
  --input "{\"conversation_id\":\"$CONVERSATION_ID\",\"status\":\"pending\"}"
```

The connector posts the selected status to the provider's `toggle_status` route and constructs an output containing the conversation ID and status.

This mutation advertises safe retry, unlike every other Chatwoot mutation. That contract does not add an idempotency header to the provider request. Read current status before repeating a call whose response was lost.

6. Handoff when required

Skip this step if the conversation remains with its current owner.

Choose at least one effect:

- `assignee_id` to change provider assignment;
- `note` to add a private handoff note.

Example with both:

```
JSON
{
  "conversation_id": "314",
  "assignee_id": "42",
  "note": "Escalated with the approved incident context."
}
```

Submit as one high-risk Action:

```
SH
HANDOFF_ACTION=act_chatwoot_lifecycle_handoff_001
HANDOFF_KEY=chatwoot-lifecycle-handoff-314-v1

aipctl action call \
  "$AIP_URL" \
  cap:chatwoot:conversation:handoff \
  --action-id "$HANDOFF_ACTION" \
  --idempotency-key "$HANDOFF_KEY" \
  --input "{\"conversation_id\":\"$CONVERSATION_ID\",\"assignee_id\":\"42\",\"note\":\"Escalated with
the approved incident context.\"}"
```

The connector performs assignment first and private-note creation second. The two provider calls are not atomic. A note failure can leave assignment completed.

An input containing only `conversation_id` returns `handoff: true` without a provider call. Reject that no-op before approval.

7. Verify final provider state

Repeat `conversation.get` and `message.list` with new read Action IDs. Verify only the fields required by the lifecycle:

- conversation ID and configured-account ownership;
- intended status;
- intended assignee when handoff requested it;
- created message or private note;
- provider timestamps and request identity where available;
- no unexpected duplicate message or note.

The generic result can contain unknown provider fields. Preserve them, but do not make completion depend on an undocumented shape.

An inbound webhook can provide additional event evidence when ingress is configured. Synchronous composite success does not depend on receiving that webhook, and a webhook does not replace the final provider read.

Retain lifecycle evidence

Keep a bounded record for each mutation:

Evidence	Required identity
Action	Action ID, capability, tenant, actor, canonical input
Approval	Approval ID, policy hash, approver, decision, expiry
Idempotency	Key, external-account scope, input hash
Route	Connector type, version, instance, account binding
Provider	HTTP status, request ID when exposed, observation time
Result	Terminal AIP status and constructed or generic output
Verification	Final read Action IDs and checked provider fields

Do not store the Chatwoot API token, webhook secret, signing seed, or unredacted customer data merely to make the record more complete.

Recover an uncertain step

Uncertain operation	Read first	Decision
`conversation.create`	Search or list conversations using authoritative provider fields	Reuse the original settled result or escalate; do not blindly create again
`message:create`	Conversation messages and AIP Action marker where exposed	Accept the existing message or escalate; do not resend blindly
`conversation:status`	Conversation status	Accept matching state or use the safe-retry contract with the same intent
Handoff assignment	Conversation assignment	Do not repeat assignment if already applied
Handoff note	Conversation messages	Issue a separately approved note-only Action only if current state requires it

Transport, provider-temporary, and oversized-response failures can mark mutation outcome uncertain. The connector failure itself sets `retryable: false`.

No lifecycle mutation has AIP rollback. A corrective status, assignment, note, or provider-catalogue update is a new governed intent against current state.

Completion criteria

The lifecycle is complete only when:

- every required capability was admitted for the intended tenant route;
- each mutation used one Action, one canonical input, and one retained key;
- each approval matched the exact mutation input;
- final provider reads confirm the intended state;
- public replies and private notes have the intended visibility;
- partial handoff effects have been resolved;
- restricted evidence is stored under tenant policy;
- no uncertain mutation was replaced with an uncorrelated retry.

These criteria establish a governed application workflow. They do not prove live-product qualification, message delivery, or production readiness.

Related documentation

- Conversation and message capabilities
- AIP composite operations
- Authentication and account isolation
- Approvals and policy
- Error reference