Chatwoot AIP composite operations

Use these three connector-owned capabilities when the strict message, status, or handoff contract matches the task. Their colon-named IDs, inputs, outputs, and provider behavior differ from the generic dot-named catalogue.

SECTION	Connectors
TYPE	Connector
PROTOCOL	AIP 1.0
SOURCE	docs/connectors/chatwoot/capabilities/aip-composite-operations.md

Read the pinned implementation caveats before calling them. Two published schemas permit inputs that do not produce the intuitive runtime behavior unless the application adds a stricter check.

Choose composite or catalogue

Composite on this page	Related catalogue capability	Choose the composite when
`cap:chatwoot:message:create`	`cap:chatwoot:message.create`	Text and public/private mode fit the strict connector input
`cap:chatwoot:conversation:status`	`cap:chatwoot:conversation.toggle_status`	The desired status is exactly `open`, `resolved`, or `pending`
`cap:chatwoot:conversation:handoff`	`cap:chatwoot:conversation.assignment` plus optional message operation	One governed intent assigns and optionally adds a private note

Colons and dots are significant. Do not normalize either ID or reuse approval, Action, or idempotency evidence across them.

Composite summary

Capability	Kind	Risk	Approval	Retry	Provider calls
`message:create`	Tool	Medium	Required	Unsafe	1
`conversation:status`	Tool	Medium	Required	Safe	1
`conversation:handoff`	Tool	High	Required	Unsafe	0–2

All three execute synchronously. None advertises async, streaming, cancellation, transaction, reconciliation, or rollback support.

Discovery and operation allowlist

The connector builds discovery by adding these three composites first and then the admitted dot-named catalogue operations. The Chatwoot operations file filters the 150-operation catalogue; it does not remove these composites from the connector-generated manifest.

Tenant catalogue admission, routing, policy, and authorization still apply. A connector-generated capability is not permission for every actor or tenant to invoke it.

Send a message

`cap:chatwoot:message:create` publishes this input:

Field	Type	Published requirement	Meaning
`conversation_id`	String	Required	Conversation in the configured account
`content`	String	Required	Message or private-note text
`private`	Boolean	Optional; schema default `false`	`false` for public reply, `true` for private note

Additional properties are rejected by the published schema. The caller cannot supply the internal AIP action marker.

Supply private explicitly

The published schema marks `private` optional, but the pinned invocation path deserializes into a Boolean field without a serde default. A direct input that omits `private` can fail as `connector.chatwoot.invalid_action`.

Send the field explicitly until source and schema agree:

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

This guidance avoids relying on a JSON Schema default as if it were applied by the connector.

Provider request

The connector posts to:

```
TEXT
POST /api/v1/accounts/{configured_account_id}/conversations/314/messages
```

It constructs a provider body with:

- the supplied `content` and `private` values;
- `content_attributes.aip.connector_id` set to `chatwoot`;
- `content_attributes.aip.loop_prevention` set to `true`;
- `content_attributes.aip.action_id` set from the real AIP Action ID.

Those attributes let inbound mapping suppress the connector's own message echo. They do not prove external delivery or provider deduplication.

Constructed result

After a successful provider response, the connector discards the provider body and returns:

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

`sent: true` means the provider request returned a successful HTTP status. It does not prove that a customer channel delivered or displayed the message.

Update conversation status

`cap:chatwoot:conversation:status` requires exactly:

Field	Type	Accepted values
`conversation_id`	Non-blank string	Conversation in the configured account
`status`	String enum	`open`, `resolved`, `pending`

Additional properties are rejected by the published schema.

Example:

```
JSON
{
  "conversation_id": "314",
  "status": "pending"
}
```

The connector posts:

```
TEXT
POST /api/v1/accounts/{configured_account_id}/conversations/314/toggle_status
```

with:

```
JSON
{
  "status": "pending"
}
```

After a successful provider response, it constructs:

```
JSON
{
  "conversation_id": "314",
  "status": "pending"
}
```

This is the only Chatwoot mutation whose connector contract advertises safe retry. The provider call still receives no AIP idempotency header. Read current conversation state before a repeat and preserve the same governed intent.

Handoff a conversation

`cap:chatwoot:conversation:handoff` publishes:

Field	Type	Published requirement	Effect
`conversation_id`	String	Required	Selects the conversation
`assignee_id`	String	Optional	Requests provider assignment
`note`	String	Optional	Sends a private handoff note

Additional properties are rejected by the schema.

Require an intent field

The published schema requires only `conversation_id`. The pinned handler does not reject an input with neither `assignee_id` nor `note`. It makes no provider request and still returns `handoff: true`.

Applications should require at least one non-empty intent field before approval:

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

This application check is stricter than the pinned capability schema. It prevents a successful no-op from being mistaken for a handoff.

Provider sequence

When `assignee_id` is present, the connector first posts:

```
TEXT
POST /api/v1/accounts/{configured_account_id}/conversations/314/assignments
```

with `{"assignee_id":"42"}`.

When `note` is present, the connector then posts a private message through the same message helper used by `message:create`. The note includes the AIP loop-prevention, connector, and Action markers.

After all requested calls succeed, the connector constructs:

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

Partial-effect boundary

Requested fields	Provider sequence	Important failure boundary
Assignee only	Assignment	A lost response can hide a completed assignment
Note only	Private message	A lost response can hide a created note
Assignee and note	Assignment, then private message	Note failure can leave assignment completed
Neither	No provider request	Connector returns success without a handoff effect

The two-call path is not atomic. It has no transaction, reconcile method, or compensating capability. Recovery must inspect assignment and message state separately.

Shared approval and idempotency contract

All three composites declare:

Contract field	Value
Approval	Required; tenant-policy selector
Approval validity	900,000 ms
Approval evidence	Reason, input snapshot, policy decision
Idempotency	Required; external-account scope; 86,400,000 ms published TTL
Data	Restricted; contains PII; redaction required
Execution	Synchronous; no async, streaming, or cancellation support
Compensation	Rollback not supported
Expected latency hint	3,000 ms
Provider timeout hint	30,000 ms
Maximum queue-delay hint	5,000 ms

The AIP idempotency key remains part of governed execution, but these composite provider requests do not add an `Idempotency-Key` header. They also do not add the catalogue's `X-AIP-Action-ID` header.

The pinned composite invoke branches do not call the catalogue's `required_idempotency_key` check. Use the governed AIP execution path, which owns contract enforcement; do not treat direct connector invocation as an idempotency gate.

Message and handoff notes embed the Action ID inside provider content attributes. Status and assignment requests contain no equivalent Action field in their constructed bodies.

Do not treat the AIP key as proof of Chatwoot deduplication. Preserve the Action and key for AIP settlement, then reconcile provider state before another call.

Side effects

Composite	Declared side effects
`message:create`	`send_message`, `write`, `external_network`
`conversation:status`	`write`, `external_network`
`conversation:handoff`	`write`, `send_message`, `identity`, `external_network`

Handoff declares the broadest policy signal because it can change ownership and add a private message. Approval should match the actual optional fields, not only the capability ID.

Failures and recovery

Failure code	Typical cause	Safe response
`connector.chatwoot.invalid_action`	Schema/runtime mismatch, invalid status, or malformed strict input	Correct input before any provider dispatch
`connector.chatwoot.authentication`	Provider returned `401` or `403`	Verify account route and credential revision
`connector.chatwoot.remote_rejected`	Provider rejected input or current state	Correct authoritative state before a new Action
`connector.chatwoot.remote_temporary`	Provider returned `429` or a server error	Read provider state; use status retry only within its contract
`connector.chatwoot.transport`	Provider exchange failed	Treat the mutation outcome as uncertain
`connector.chatwoot.response_too_large`	Provider response crossed the active bound	Reconcile state; do not enlarge the bound to force a repeat

Every failure sets `retryable: false` and omits a provider request ID. The status capability's safe-retry contract is a separate policy signal.

For uncertain message creation, inspect the conversation for the Action marker before another call. For uncertain status, read the conversation and compare the intended state. For uncertain handoff, inspect assignment first and then the private note.

If assignment completed but note failed, do not repeat the entire handoff blindly. Decide whether a separately approved note-only Action is appropriate for current state.

Verify an integration

Require all of the following:

- discovery returns the exact colon-named capability;
- the operations allowlist is not mistaken for composite authorization;
- `private` is present explicitly for `message:create`;
- handoff contains at least one reviewed intent field;
- the governed AIP path enforces the required idempotency contract;
- approval binds every optional field and the intended external effect;
- one Action ID, canonical input, and idempotency key are retained;
- constructed output is not mistaken for raw provider output;
- message delivery, status, assignment, and note state are verified through

authoritative Chatwoot reads;

- loop-prevention attributes are retained for connector-originated messages;
- partial or uncertain outcomes are reconciled instead of repeated.

These checks validate application handling. They do not establish customer delivery, provider deduplication, atomic handoff, or live qualification.

Related documentation

- Chatwoot capability index
- Conversations and messages
- Run the Chatwoot connector quickstart
- Approvals and policy
- Error reference