Chatwoot conversations and messages

Use these 28 provider-catalogue capabilities to find conversations, change their state, manage routing metadata, and work with messages. Start by choosing the exact capability ID; three similar connector composites use different punctuation

SECTION	Connectors
TYPE	Connector
PROTOCOL	AIP 1.0
SOURCE	docs/connectors/chatwoot/capabilities/conversations-and-messages.md

The connector inserts its configured Chatwoot account. Applications supply only conversation, message, query, and provider-body values allowed by the selected operation.

Family at a glance

Group	Operations	Reads	Mutations	High risk
Core conversation records	9	5	4	1
State, routing, and metadata	13	3	10	0
Messages	6	1	5	1
Total	**28**	**9**	**19**	**2**

The method split is nine `GET`, 15 `POST`, two `PATCH`, and two `DELETE` operations. The nine reads are low risk, 17 mutations are medium risk, and the two deletes are high risk.

Twenty-seven capabilities are Tools. `conversation.transcript` is a Workflow, but its execution contract is still synchronous and has no async, streaming, or cancellation support.

Distinguish catalogue operations from composites

Provider-catalogue capability on this page	Similar connector composite	Important difference
`cap:chatwoot:message.create`	`cap:chatwoot:message:create`	Dot ID uses generic provider input and result; colon ID has strict connector-owned message fields
`cap:chatwoot:conversation.toggle_status`	`cap:chatwoot:conversation:status`	Dot ID passes a generic provider body; colon ID accepts a strict status enum
`cap:chatwoot:conversation.assignment`	`cap:chatwoot:conversation:handoff`	Dot ID calls the assignment route; colon ID can assign and add a private note

Do not normalize dots and colons. They are distinct capability IDs, and an approval or idempotency record for one does not authorize another.

Core conversation operations

Capability	Provider request	Purpose	Risk	Approval
`cap:chatwoot:conversation.list`	`GET /api/v1/accounts/{account_id}/conversations`	List conversations	Low	No
`cap:chatwoot:conversation.create`	`POST /api/v1/accounts/{account_id}/conversations`	Create a conversation	Medium	Required
`cap:chatwoot:conversation.get`	`GET /api/v1/accounts/{account_id}/conversations/{conversation_id}`	Read one conversation	Low	No
`cap:chatwoot:conversation.update`	`PATCH /api/v1/accounts/{account_id}/conversations/{conversation_id}`	Update one conversation	Medium	Required
`cap:chatwoot:conversation.delete`	`DELETE /api/v1/accounts/{account_id}/conversations/{conversation_id}`	Delete one conversation	High	Required
`cap:chatwoot:conversation.meta`	`GET /api/v1/accounts/{account_id}/conversations/meta`	Read counters and metadata	Low	No
`cap:chatwoot:conversation.search`	`GET /api/v1/accounts/{account_id}/conversations/search`	Search conversations	Low	No
`cap:chatwoot:conversation.unread_counts`	`GET /api/v1/accounts/{account_id}/conversations/unread_counts`	Read unread counts	Low	No
`cap:chatwoot:conversation.filter`	`POST /api/v1/accounts/{account_id}/conversations/filter`	Filter conversations	Medium	Required

`conversation.filter` has a read-like purpose but uses `POST`. The frozen catalogue classifies every non-`GET` operation as a mutation. It therefore requires approval and an idempotency key and does not advertise retry support.

State, routing, and metadata operations

Capability	Provider request	Purpose	Risk	Approval
`cap:chatwoot:conversation.mute`	`POST /api/v1/accounts/{account_id}/conversations/{conversation_id}/mute`	Mute a conversation	Medium	Required
`cap:chatwoot:conversation.unmute`	`POST /api/v1/accounts/{account_id}/conversations/{conversation_id}/unmute`	Unmute a conversation	Medium	Required
`cap:chatwoot:conversation.transcript`	`POST /api/v1/accounts/{account_id}/conversations/{conversation_id}/transcript`	Request a transcript	Medium	Required
`cap:chatwoot:conversation.priority`	`POST /api/v1/accounts/{account_id}/conversations/{conversation_id}/toggle_priority`	Update priority	Medium	Required
`cap:chatwoot:conversation.unread`	`POST /api/v1/accounts/{account_id}/conversations/{conversation_id}/unread`	Mark unread	Medium	Required
`cap:chatwoot:conversation.custom_attributes`	`POST /api/v1/accounts/{account_id}/conversations/{conversation_id}/custom_attributes`	Update custom attributes	Medium	Required
`cap:chatwoot:conversation.attachment.list`	`GET /api/v1/accounts/{account_id}/conversations/{conversation_id}/attachments`	List attachments	Low	No
`cap:chatwoot:conversation.assignment`	`POST /api/v1/accounts/{account_id}/conversations/{conversation_id}/assignments`	Assign an agent or team	Medium	Required
`cap:chatwoot:conversation.label.list`	`GET /api/v1/accounts/{account_id}/conversations/{conversation_id}/labels`	List applied labels	Low	No
`cap:chatwoot:conversation.label.update`	`POST /api/v1/accounts/{account_id}/conversations/{conversation_id}/labels`	Replace applied labels	Medium	Required
`cap:chatwoot:conversation.toggle_status`	`POST /api/v1/accounts/{account_id}/conversations/{conversation_id}/toggle_status`	Toggle provider status	Medium	Required
`cap:chatwoot:conversation.toggle_typing`	`POST /api/v1/accounts/{account_id}/conversations/{conversation_id}/toggle_typing_status`	Publish typing state	Medium	Required
`cap:chatwoot:conversation.reporting_event.list`	`GET /api/v1/accounts/{account_id}/conversations/{conversation_id}/reporting_events`	List enterprise reporting events	Low	No

The word `list` in `conversation.reporting_event.list` matches its `GET` contract. The word `transcript` does not imply a background AIP run; this Workflow completes through the same synchronous catalogue path.

Typing state, labels, priority, assignment, and custom attributes can affect customer-service workflows. The generic contract marks only `write` and `external_network`, so policy must also inspect the exact capability and body.

Message operations

Capability	Provider request	Purpose	Risk	Approval
`cap:chatwoot:message.list`	`GET /api/v1/accounts/{account_id}/conversations/{conversation_id}/messages`	List messages	Low	No
`cap:chatwoot:message.update`	`PATCH /api/v1/accounts/{account_id}/conversations/{conversation_id}/messages/{message_id}`	Update one message	Medium	Required
`cap:chatwoot:message.delete`	`DELETE /api/v1/accounts/{account_id}/conversations/{conversation_id}/messages/{message_id}`	Delete one message	High	Required
`cap:chatwoot:message.translate`	`POST /api/v1/accounts/{account_id}/conversations/{conversation_id}/messages/{message_id}/translate`	Translate one message	Medium	Required
`cap:chatwoot:message.retry`	`POST /api/v1/accounts/{account_id}/conversations/{conversation_id}/messages/{message_id}/retry`	Ask Chatwoot to retry a failed outgoing message	Medium	Required
`cap:chatwoot:message.create`	`POST /api/v1/accounts/{account_id}/conversations/{conversation_id}/messages`	Create a message with optional attachments	Medium	Required

`message.retry` is a provider operation, not permission for AIP to repeat the Action. Its own capability contract declares retry unsafe. Repeating it after an uncertain result can ask Chatwoot to process the failed message again.

Account and path inputs

The host inserts `account_id`. It is not a caller input.

`conversation_id` is required by 22 operations. It is omitted only for:

- `conversation.list`;
- `conversation.create`;
- `conversation.meta`;
- `conversation.search`;
- `conversation.unread_counts`;
- `conversation.filter`.

`message_id` is additionally required by `message.update`, `message.delete`, `message.translate`, and `message.retry`.

Every path input is a non-blank string of at most 512 bytes without control characters. Each value becomes one encoded URL segment.

Generic input envelope

The catalogue adds optional `query`, `body`, and `multipart` properties around the required path IDs:

Property	Connector boundary
`query`	Object with at most 128 keys; scalar values or arrays of at most 256 scalars

Property	Connector boundary
`body`	Object with at most 512 properties and a 16 MiB encoded limit
`multipart.fields`	Object with at most 512 fields
`multipart.files`	Array with at most 20 file objects

`body` and `multipart` are mutually exclusive. Provider-specific message, conversation, filter, assignment, label, status, and translation fields are not frozen by this generic schema.

An input can satisfy the AIP schema and still be rejected by Chatwoot. Use the payload contract for the exact pinned provider route, then retain its canonical form with approval evidence.

Read one conversation

Use a conversation ID obtained from the same account route:

```
SH
aipctl action call \
  "$AIP_URL" \
  cap:chatwoot:conversation.get \
  --action-id act_chatwoot_conversation_get_001 \
  --input '{"conversation_id":"314"}'
```

The connector sends:

```
TEXT
GET /api/v1/accounts/{configured_account_id}/conversations/314
```

This read needs no capability-level approval and advertises safe retry. Its result can contain customer and message metadata, so the restricted-data and redaction contract still applies.

Create a message with an attachment

The pinned connector fixture exercises `cap:chatwoot:message.create` with this exact input shape:

```
JSON
{
  "conversation_id": "conversation-1",
  "multipart": {
    "fields": {
      "content": "Evidence attached",
      "private": false
    },
    "files": [
      {
        "field_name": "attachments[]",
        "filename": "evidence.txt",
        "content_type": "text/plain",
        "content_base64": "aGVsbG8="
      }
    ]
  }
}
```

The fixture proves connector-side multipart encoding and allowlist enforcement. It does not prove that an unidentified Chatwoot version accepts the same payload.

Multipart bounds

Value	Hard connector bound
Files per request	20
Decoded bytes per file	25 MiB
Combined field-value and decoded-file bytes	64 MiB
Serialized bytes per field value	1 MiB
Field name	1–128 bytes; no control characters
Filename	1–255 bytes; basename only
Content type	1–255 bytes

Each file object must contain exactly `field_name`, `filename`, `content_type`, and standard-base64 `content_base64`. An empty multipart object is rejected.

Submit the message mutation with a unique Action ID, exact approval, and one external-account-scoped idempotency key. The connector forwards `X-AIP-Action-ID` and `Idempotency-Key` to the provider.

Shared mutation contract

All 19 non-`GET` operations 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
Connector retry	Not supported; unsafe
Side effects	`write`, `external_network`
Data	Restricted; contains PII; redaction required
Execution	Synchronous; no async, streaming, or cancellation support
Compensation	Rollback not supported

The connector forwards an idempotency key but does not prove provider deduplication. Preserve one key only for the same capability, account, Action, path IDs, and canonical input.

Result contract

A successful catalogue operation returns:

```
JSON
{
  "http_status": 200,
  "body": {},
  "provider_request_id": null
}
```

`body` contains parsed JSON, `null` for an empty provider body, or a `text` wrapper for non-JSON bytes. `provider_request_id` reads `x-request-id` first and `x-runtime` second.

This generic result differs from the strict outputs of the three colon-named composites. Do not deserialize one as the other.

Failures and recovery

Failure code	Typical cause	Safe response
`connector.chatwoot.invalid_action`	Unknown or unadmitted ID, missing path input, conflicting body formats, or malformed multipart	Correct input or refresh discovery before dispatch
`connector.chatwoot.missing_idempotency_key`	Non-`GET` Action has no valid key	Restore the key for the original intent
`connector.chatwoot.authentication`	Provider returned `401` or `403`	Verify account binding and credential revision
`connector.chatwoot.remote_rejected`	Chatwoot rejected input or state	Correct authoritative state before another Action
`connector.chatwoot.remote_temporary`	Provider returned `429` or a server error	Retry a read within policy; reconcile a mutation
`connector.chatwoot.transport`	Provider exchange failed	Retry a read within policy; treat a mutation as uncertain
`connector.chatwoot.response_too_large`	Response exceeded the active bound	Preserve evidence and reconcile a mutation

Connector failures set `retryable: false`. Only the nine read contracts advertise retry support.

For an uncertain create or update, search or read the target before submitting a correction. For an uncertain delete, compare authoritative state and audit evidence. For an uncertain `message.retry` or `message.create`, inspect the conversation before risking duplicate customer-visible delivery.

Approval and idempotency do not provide rollback. Corrective writes require a new decision based on current Chatwoot state.

Verify an integration

Require all of the following:

- discovery returns the exact dot-named catalogue capability;
- the Action does not contain a caller-selected account ID;
- conversation and message IDs belong to the configured account;
- provider body fields match the pinned route;
- every non-`GET` operation has exact approval and one retained key;
- multipart content stays within every decoded and aggregate bound;
- the completed result belongs to the original Action and preserves provider

status and request identity;

- consequential mutations are confirmed through authoritative conversation

state;

- restricted conversation, message, transcript, and attachment data is

redacted and retained under tenant policy;

- uncertain mutations are reconciled instead of repeated.

These checks validate application behavior. They do not establish live Chatwoot qualification or customer-delivery correctness.

Related documentation

- Chatwoot capability index
- Accounts, agents, teams, and bots

- Run the Chatwoot connector quickstart
- Approvals and policy
- Error reference