Chatwoot inboxes, integrations, and webhooks

Use these 21 capabilities to inspect and configure Chatwoot inboxes, membership, integration hooks, and account webhook records. Select the family that owns the provider object; similarly named operations do not configure the same boundary.

SECTION	Connectors
TYPE	Connector
PROTOCOL	AIP 1.0
SOURCE	docs/connectors/chatwoot/capabilities/inboxes-integrations-and-webhooks.md

These capabilities call Chatwoot's account API. They are separate from the connector host's inbound `/webhooks/chatwoot` route, its HMAC secret, and its durable replay fence.

Family at a glance

Group	Operations	Reads	Mutations	High risk
Inboxes and members	13	5	8	3
Integration apps and hooks	4	1	3	1
Account webhooks	4	1	3	1
Total	**21**	**7**	**14**	**5**

All 21 capabilities are Tools. The seven `GET` operations are low-risk reads. Nine ordinary `POST` and `PATCH` operations are medium risk. The `inbox.webhook.register` operation and all four deletes are high risk.

All 14 mutations require approval and an idempotency key. None advertises connector retry support or compensation.

Choose the correct surface

Task	Capability group	Boundary
Inspect or configure a channel	`inbox.*`	Inbox object in the configured account
Read or replace inbox membership	`inbox.member.*`	Provider inbox-member collection
Inspect provider integration applications	`integration.app.list`	Account integration-app inventory
Create, update, or remove an integration hook	`integration.hook.*`	Provider integration hook
Create, update, or remove an account webhook	`webhook.*`	Provider account-webhook object
Register the provider webhook for one inbox	`inbox.webhook.register`	Special high-risk inbox operation
Receive a signed delivery in AIP	Not an Action capability	Connector-host webhook ingress

Creating a provider webhook does not enable the connector-host ingress. The host route exists only when its owner supplies the webhook secret file and the required event-publication endpoint.

Inbox and member operations

Capability	Provider request	Purpose	Risk	Approval
`cap:chatwoot:inbox.list`	`GET /api/v1/accounts/{account_id}/inboxes`	List inboxes	Low	No
`cap:chatwoot:inbox.create`	`POST /api/v1/accounts/{account_id}/inboxes`	Create an inbox	Medium	Required
`cap:chatwoot:inbox.get`	`GET /api/v1/accounts/{account_id}/inboxes/{inbox_id}`	Read one inbox	Low	No
`cap:chatwoot:inbox.update`	`PATCH /api/v1/accounts/{account_id}/inboxes/{inbox_id}`	Update one inbox	Medium	Required
`cap:chatwoot:inbox.delete`	`DELETE /api/v1/accounts/{account_id}/inboxes/{inbox_id}`	Delete one inbox	High	Required
`cap:chatwoot:inbox.agent.list`	`GET /api/v1/accounts/{account_id}/inboxes/{inbox_id}/assignable_agents`	List assignable agents	Low	No
`cap:chatwoot:inbox.health`	`GET /api/v1/accounts/{account_id}/inboxes/{inbox_id}/health`	Read provider inbox health	Low	No
`cap:chatwoot:inbox.template.sync`	`POST /api/v1/accounts/{account_id}/inboxes/{inbox_id}/sync_templates`	Synchronize inbox templates	Medium	Required
`cap:chatwoot:inbox.webhook.register`	`POST /api/v1/accounts/{account_id}/inboxes/{inbox_id}/register_webhook`	Register the provider webhook for an inbox	High	Required
`cap:chatwoot:inbox.member.list`	`GET /api/v1/accounts/{account_id}/inbox_members/{inbox_id}`	Read inbox members	Low	No
`cap:chatwoot:inbox.member.create`	`POST /api/v1/accounts/{account_id}/inbox_members`	Add inbox members	Medium	Required
`cap:chatwoot:inbox.member.update`	`PATCH /api/v1/accounts/{account_id}/inbox_members`	Replace inbox members	Medium	Required
`cap:chatwoot:inbox.member.delete`	`DELETE /api/v1/accounts/{account_id}/inbox_members`	Remove inbox members	High	Required

The three member mutations use a collection route without an `inbox_id` path placeholder. Their provider body must identify the intended membership according to the pinned Chatwoot API. The generic AIP schema does not enforce that provider rule.

`inbox.health` reads provider state for one channel. It does not report connector-host readiness, registry admission, webhook replay storage, or event publication health.

Integration operations

Capability	Provider request	Purpose	Risk	Approval
`cap:chatwoot:integration.app.list`	`GET /api/v1/accounts/{account_id}/integrations/apps`	List integration applications	Low	No
`cap:chatwoot:integration.hook.create`	`POST /api/v1/accounts/{account_id}/integrations/hooks`	Create an integration hook	Medium	Required
`cap:chatwoot:integration.hook.update`	`PATCH /api/v1/accounts/{account_id}/integrations/hooks/{hook_id}`	Update one integration hook	Medium	Required
`cap:chatwoot:integration.hook.delete`	`DELETE /api/v1/accounts/{account_id}/integrations/hooks/{hook_id}`	Delete one integration hook	High	Required

The reviewed surface can list integration applications, but it cannot create, update, or delete an application. The three mutations operate only on integration hooks.

Account-webhook operations

Capability	Provider request	Purpose	Risk	Approval
`cap:chatwoot:webhook.list`	`GET /api/v1/accounts/{account_id}/webhooks`	List account webhooks	Low	No
`cap:chatwoot:webhook.create`	`POST /api/v1/accounts/{account_id}/webhooks`	Create an account webhook	Medium	Required
`cap:chatwoot:webhook.update`	`PATCH /api/v1/accounts/{account_id}/webhooks/{webhook_id}`	Update one account webhook	Medium	Required
`cap:chatwoot:webhook.delete`	`DELETE /api/v1/accounts/{account_id}/webhooks/{webhook_id}`	Delete one account webhook	High	Required

These operations manage provider records. They do not write `AIP_CHATWOOT_WEBHOOK_SECRET_FILE`, create an AIP event-publication route, or prove that Chatwoot can reach the connector host.

Account and path inputs

The connector inserts its configured `account_id` into every path. Action input never selects the account.

Input	Required by
`inbox_id`	`inbox.get`, `inbox.update`, `inbox.delete`, `inbox.agent.list`, `inbox.health`, `inbox.template.sync`, `inbox.webhook.register`, and `inbox.member.list`
`hook_id`	`integration.hook.update` and `integration.hook.delete`
`webhook_id`	`webhook.update` and `webhook.delete`

Each path input is a non-blank string of at most 512 bytes and cannot contain a control character. The URL builder encodes each value as one path segment.

The common catalogue envelope also accepts:

- an optional `query` object with at most 128 keys;
- an optional `body` object with at most 512 properties and a 16 MiB encoded

request limit;

- an optional bounded `multipart` object instead of `body`.

The connector rejects an input containing both `body` and `multipart`. It does not freeze provider-specific body fields for any of these 21 operations. Validate every body against the exact provider route and pinned Chatwoot revision before requesting approval.

Read inbox health

Use the inbox ID from an admitted provider record:

```
SH
aipctl action call \
  "$AIP_URL" \
  cap:chatwoot:inbox.health \
  --action-id act_chatwoot_inbox_health_001 \
  --input '{"inbox_id":"7"}'
```

The connector calls:

```
TEXT
GET /api/v1/accounts/{configured_account_id}/inboxes/7/health
```

The operation is a low-risk read and is safe to retry within the Action contract. Its provider body remains restricted data and may contain channel detail. Do not expose it merely because the operation needs no human approval.

Prepare a configuration mutation

The pinned Rust catalogue deliberately does not own provider payload fields. For `webhook.create`, `integration.hook.create`, or `inbox.webhook.register`, use this sequence:

1. Confirm that tenant discovery admits the exact capability.
2. Select the payload schema for the pinned Chatwoot route.
3. Build one bounded `body` object and retain its canonical form.
4. Bind approval to the exact capability, body, actor, tenant, and Action.
5. Assign one external-account-scoped idempotency key.
6. Dispatch the Action once.
7. Verify provider state with the corresponding read when one exists.

No copy-ready provider body appears here because the connector source does not validate one. Adding guessed URL, subscription, or integration fields would turn a source-backed reference into an unpinned provider example.

`inbox.webhook.register` is high risk even though it uses `POST`. Treat the destination and resulting delivery path as security-sensitive configuration. Approval does not prove endpoint ownership, reachability, or signature compatibility.

Keep provider management and inbound delivery separate

Boundary	Owned by	Source-visible control
Provider webhook or integration record	A Chatwoot capability	Approved provider mutation
Connector inbound route	`aip-host-chatwoot`	`POST /webhooks/chatwoot`
Inbound HMAC material	Connector host operator	Owner-only webhook secret file
Delivery freshness and replay	Connector runtime	300-second timestamp window and durable instance-scoped replay state
Event publication	Connector host and runtime	Configured connector-host event endpoint and durable outbox
Loop prevention	Connector mapping	AIP content attributes on connector-originated messages

Omitting the webhook secret file creates an outbound-only host and no Chatwoot webhook route. Supplying the file also requires the connector-host event endpoint; startup rejects the incomplete ingress configuration.

Detailed signature construction, required headers, replay behavior, loop prevention, and delivery errors belong to `webhook-security-loop-prevention-and-replay.md` later in this connector section.

Shared mutation contract

All 14 mutations 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
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

Provider configuration may cause later network traffic or automated behavior, but the generic contracts do not enumerate those downstream effects. Policy must consider the selected operation and body, not only the two generic side-effect markers.

Result contract

Every successful catalogue operation returns:

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

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

The output schema does not define provider webhook, inbox, or integration fields. Validate the shape needed by the application and retain unknown fields for compatibility.

Failures and recovery

Failure code	Typical cause	Safe response
`connector.chatwoot.invalid_action`	Unknown or unadmitted operation, missing path input, or malformed generic envelope	Refresh discovery or correct input before dispatch
`connector.chatwoot.missing_idempotency_key`	Mutation has no valid key	Restore the key bound to the same canonical intent
`connector.chatwoot.authentication`	Provider returned `401` or `403`	Verify account binding and credential revision
`connector.chatwoot.remote_rejected`	Provider rejected the body or current state	Correct the payload or provider state before a new 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 mutation outcome as uncertain
`connector.chatwoot.response_too_large`	Response crossed the configured bound	Inspect provider output and reconcile a mutation

All connector failures advertise `retryable: false`. The seven read contracts separately allow a bounded retry. Mutations remain unsafe to repeat even when they carry an idempotency key.

After an uncertain provider-configuration mutation, inspect Chatwoot before creating another Action. A lost response can hide a successfully created hook or inbox. Repeating it with a new identity can create duplicate delivery paths.

Verify an integration

Require all of the following:

- the selected capability matches the provider object being managed;
- discovery admits the exact suffix for the intended account route;
- no caller-selected account ID crosses the host boundary;
- path IDs and provider body come from authoritative state;
- every mutation has exact approval evidence and one retained idempotency key;
- a provider read confirms the resulting record when available;
- connector-host ingress is configured and verified independently;
- restricted payloads, destinations, tokens, and event data are redacted;
- uncertain mutations are reconciled instead of repeated.

These checks validate application handling. They do not prove provider delivery, inbound signature compatibility, or live qualification.

Related documentation

- Chatwoot capability index
- Chatwoot connector overview
- Chatwoot connector configuration
- Authentication and account isolation
- Approvals and policy