Cal.diy slots and reservations

Use these capabilities to query availability and to create, inspect, update, or release a short-lived slot reservation. The reviewed connector exposes five fixed operations, all pinned to Cal.diy API version 2024-09-04.

SECTION	Connectors
TYPE	Connector
PROTOCOL	AIP 1.0
SOURCE	docs/connectors/cal-diy/capabilities/slots-and-reservations.md

A slot result is a time-bounded observation. It does not reserve capacity and does not guarantee that a later booking will commit. Use a reservation when the workflow needs a provider-owned hold, and preserve its opaque uid.

Operations

Capability	Provider request	Risk	Approval
`cap:cal_diy:slot.list`	`GET /v2/slots`	Low	No
`cap:cal_diy:slot.reservation.create`	`POST /v2/slots/reservations`	Medium	Required
`cap:cal_diy:slot.reservation.get`	`GET /v2/slots/reservations/{reservation_uid}`	Low	No
`cap:cal_diy:slot.reservation.update`	`PATCH /v2/slots/reservations/{reservation_uid}`	Medium	Required
`cap:cal_diy:slot.reservation.delete`	`DELETE /v2/slots/reservations/{reservation_uid}`	High	Required

Applications invoke the AIP capability and pass one input object. The connector encodes the slot query, percent-encodes the reservation path, and removes path fields from provider request bodies.

Query available slots

`cap:cal_diy:slot.list` requires `start`, `end`, and exactly one selector branch. Both boundaries accept an ISO date or date-time.

Select by event-type id

```
JSON
{
  "start": "2030-01-01",
  "end": "2030-01-08",
  "eventTypeId": 42,
  "timeZone": "Europe/London",
  "format": "time"
}
```

`eventTypeId` is an integer.

Select by user and event slug

```
JSON
{
  "start": "2030-01-01T00:00:00Z",
  "end": "2030-01-08T00:00:00Z",
  "eventTypeSlug": "product-consultation",
  "username": "avery"
}
```

`eventTypeSlug` and `username` must be non-empty. `organizationSlug` is optional in this branch.

Select by team and event slug

```
JSON
{
  "start": "2030-01-01",
  "end": "2030-01-08",
  "eventTypeSlug": "team-consultation",
  "teamSlug": "solutions"
}
```

`eventTypeSlug` and `teamSlug` must be non-empty. `organizationSlug` is optional in this branch.

Select a dynamic group

```
JSON
{
  "start": "2030-01-01",
  "end": "2030-01-08",
  "usernames": ["avery", "jordan"],
  "organizationSlug": "example-org"
}
```

This branch requires at least two unique, non-empty usernames and a non-empty organization slug.

The four branches reject each other's selector fields. For example, do not add `teamSlug` to an `eventTypeId` query or combine `username` with `usernames`. Additional properties are rejected.

Shared query fields

Field	Constraint	Purpose
`start`	ISO date or date-time; required	Beginning of the query interval
`end`	ISO date or date-time; required	End of the query interval
`timeZone`	Non-empty string	Requested time-zone interpretation
`duration`	Integer at least `1`	Requested slot duration
`bookingUidToReschedule`	Non-empty string	Exclude or evaluate an existing booking during reschedule
`format`	`time` or `range`	Provider response representation

The published schema validates field shapes, not chronological ordering or time-zone identifiers. Cal.diy can reject a schema-valid interval or selector under provider-side policy.

Interpret slot results

Slot discovery is a public-data read in the capability contract. It contains no declared PII and does not require redaction. Deployment policy may still limit which account, event type, team, or date range a caller may query.

The operation is synchronous, retry-safe, and eligible for connector retry. It uses optional action-scoped idempotency with input-hash revalidation. Retry only within the caller deadline and current policy; a later response may legitimately show different availability.

The connector returns the provider JSON body after checking the common output schema. A non-empty string `status` is required. The `data`, `pagination`, and any additional provider fields are provider-owned and optional in the published connector schema.

Create a reservation

`cap:cal_diy:slot.reservation.create` requires an event-type id and a slot start:

```
JSON
{
  "eventTypeId": 42,
  "slotStart": "2030-01-03T10:00:00Z",
  "slotDuration": 30,
  "reservationDuration": 5
}
```

Field	Constraint
`eventTypeId`	Integer; required
`slotStart`	Date-time string; required
`slotDuration`	Integer at least `1`
`reservationDuration`	Integer at least `1`

Additional properties are rejected before provider dispatch.

The published JSON Schema does not encode a unit for the two duration fields. The source-owned ignored live exercise names its slot-duration input in minutes and uses `5` for `reservationDuration`. Treat that as test-source context, not as evidence that a live run passed or that a future provider version preserves the same interpretation.

Create is a medium-risk mutation. It requires approval and an external-account-scoped idempotency key, but it does not perform a separate slot-list preflight. A schema-valid request can still lose an availability race or be rejected by Cal.diy.

Capture and read the reservation

The source-owned ignored live exercise expects a successful create response to contain a non-empty string at `/data/reservationUid`, then uses that value for release. The connector's common output schema does not require that field.

Validate the returned uid before storing or using it. If it is absent, stop and treat the response as provider-contract drift; do not guess an identifier from the action id or idempotency key.

`cap:cal_diy:slot.reservation.get` accepts only:

```
JSON
{
  "reservation_uid": "reservation-provider-uid"
}
```

`reservation_uid` must be a non-empty string. It is a sensitive provider handle even though the capability's data contract does not classify reservation operations as containing PII. Do not place it in application logs, user-visible URLs, or metric labels. The connector necessarily places its percent-encoded value in the provider request path.

Reservation get is a low-risk, retry-safe read. It does not extend or recreate the reservation.

Update the reservation

Update requires the existing uid, event-type id, and new slot start:

```
JSON
{
  "reservation_uid": "reservation-provider-uid",
```

```
  "eventTypeId": 42,
  "slotStart": "2030-01-03T10:30:00Z",
  "slotDuration": 30,
  "reservationDuration": 5
}
```

`slotDuration` and `reservationDuration` remain optional positive integers. All other properties are rejected.

Before update dispatch, the connector reads the current reservation by uid. That preflight checks current provider visibility; it does not reserve the new time, guarantee the update, or make the mutation retry-safe.

Release the reservation

Release uses the delete capability with the same one-field input as get:

```
JSON
{
  "reservation_uid": "reservation-provider-uid"
}
```

This is a high-risk destructive mutation. The connector reads the current reservation before dispatch, then sends `DELETE` to its fixed provider route. After a completed release, do not assume that the uid can be reused.

Release declares no rollback or compensation. Creating a later reservation is new provider state and may fail because availability has changed.

Mutation safety and compensation

Create, update, and release share these contract requirements:

Contract field	Value
Human approval	Required
Approval selector	Tenant policy
Approval validity window	900,000 ms
Approval evidence	Reason, input snapshot, policy decision
Idempotency	Required; external-account scope
Collision behavior	Revalidate input hash
Connector retry	Not supported
Retry safety	Unsafe
Execution	Synchronous only
Transaction modes	`dry_run`, `plan`, `commit`, `reconcile`
Plan before commit	Not required
Dry-run fidelity	Policy and schema only

Create and update use the `customer_visible_change` approval reason class. Release uses `destructive_state_change`.

Reservation create names `slot.reservation.delete` as a best-effort, approval-required compensation with a 24-hour contract window. Compensation is a new governed release action; it is not automatic and cannot guarantee that downstream workflow effects disappear.

Update and release declare rollback unsupported. If a mutation outcome is uncertain, reconcile the durable provider-operation evidence instead of sending the action again under a new key.

Reservation operations are confidential but declare no PII and set the contract's `redaction_required` marker to false. The connector still applies its generic secret-name redactor to every provider body. Neither fact makes the

reservation uid public.

Failures and recovery

Failure	Relevant cause	Safe response
`connector.cal_diy.invalid_action`	Wrong selector branch, missing required field, invalid date shape, empty uid, or unknown property	Correct input before another call
`connector.cal_diy.policy`	Mutation omitted its idempotency key	Preserve the approved input and supply the required key
`connector.cal_diy.idempotency_collision`	The key already owns different input	Stop and inspect the original action
`connector.cal_diy.in_flight`	The matching reservation mutation is still running	Read durable action state and wait
`connector.cal_diy.outcome_unknown`	A prior provider dispatch cannot be proved	Reconcile; do not create a replacement key
`connector.cal_diy.authentication`	Provider returned `401` or `403`	Verify account binding and credential revision
`connector.cal_diy.state_conflict`	Provider returned `409` or `412`	Read current reservation or slots and review the intended change
`connector.cal_diy.rate_limited`	Provider returned `429`	Retry slot or reservation reads within policy; reconcile mutations
`connector.cal_diy.remote_temporary`	Provider returned a server error	Retry reads within policy; reconcile mutations
`connector.cal_diy.transport`	Provider exchange failed	Retry reads only; treat dispatched mutations as uncertain
`connector.cal_diy.invalid_provider_output`	Successful response lacked a valid `status`	Preserve evidence and reconcile a mutation

Permanent provider rejection, invalid credentials, oversized responses, and durable settlement failures use the common Cal.diy error family. Gateway and lifecycle errors are described in the global error reference.

Verify the lifecycle

For slot discovery, verify the selected branch, account route, query interval, response status, and observation time. Do not cache the result beyond the workflow's availability policy.

For a reservation lifecycle, verify that:

- discovery returns the exact reservation capability before each action;
- approval binds the exact mutation input;
- each mutation retains its external-account idempotency key;
- create returns a validated non-empty reservation uid before later calls;
- update or release reaches a completed durable state;
- a read confirms the expected reservation state when the provider supports it;
- uncertain dispatch is reconciled instead of repeated.

The ignored live exercise in source describes a reversible round trip, but its presence is not retained execution evidence. These checks do not claim live provider qualification or production readiness.

Related documentation

- Cal.diy capability index
- Authentication and tenant routing
- Approvals and policy

- Transactions and compensation
- Error reference