Cal.diy booking capabilities

Use these capabilities to find, create, reschedule, cancel, confirm, decline, mark, reassign, or relocate a Cal.diy booking. This page owns 12 core lifecycle operations. A separate capability page owns the 10 attendee, guest, calendar link,

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

All booking operations use provider API version `2024-08-13`. Booking data is restricted, contains PII, and requires redaction throughout this family.

Choose an operation

Find a booking

Capability	Provider request	Risk	Approval
`cap:cal_diy:booking.list`	`GET /v2/bookings`	Low	No
`cap:cal_diy:booking.get`	`GET /v2/bookings/{booking_uid}`	Low	No
`cap:cal_diy:booking.get_by_seat`	`GET /v2/bookings/by-seat/{seat_uid}`	Low	No

Create or move a booking

Capability	Provider request	Risk	Plan required
`cap:cal_diy:booking.create`	`POST /v2/bookings`	Medium	Yes
`cap:cal_diy:booking.reschedule`	`POST /v2/bookings/{booking_uid}/reschedule`	High	Yes
`cap:cal_diy:booking.location.update`	`PATCH /v2/bookings/{booking_uid}/location`	Medium	No

Change booking state or ownership

Capability	Provider request	Risk	Plan required
`cap:cal_diy:booking.cancel`	`POST /v2/bookings/{booking_uid}/cancel`	High	Yes
`cap:cal_diy:booking.confirm`	`POST /v2/bookings/{booking_uid}/confirm`	High	No
`cap:cal_diy:booking.decline`	`POST /v2/bookings/{booking_uid}/decline`	High	No
`cap:cal_diy:booking.mark_absent`	`POST /v2/bookings/{booking_uid}/mark-absent`	High	No
`cap:cal_diy:booking.reassign`	`POST /v2/bookings/{booking_uid}/reassign`	High	No
`cap:cal_diy:booking.reassign_to_user`	`POST /v2/bookings/{booking_uid}/reassign/{user_id}`	High	No

Every mutation requires approval and an idempotency key even when the table does not require a transaction plan.

List bookings

`cap:cal_diy:booking.list` accepts an empty object or any supported filter:

Field	Constraint
`status`	Non-empty unique array of `upcoming`, `recurring`, `past`, `cancelled`, or `unconfirmed`
`attendeeEmail`	Email string
`attendeeName`	String
`bookingUid`	Non-empty string
`eventTypeIds`	Non-empty array of integers
`eventTypeId`	Integer
`teamsIds`	Non-empty array of integers
`teamId`	Integer
`afterStart`, `beforeEnd`	Date-time strings
`afterCreatedAt`, `beforeCreatedAt`	Date-time strings
`afterUpdatedAt`, `beforeUpdatedAt`	Date-time strings
`sortStart`, `sortEnd`	`asc` or `desc`
`sortCreated`, `sortUpdatedAt`	`asc` or `desc`
`take`	Integer from `1` through `250`
`skip`	Integer at least `0`

Additional fields are rejected. The connector encodes these values as query parameters. It does not add logical constraints between overlapping singular and plural filters; Cal.diy can reject or further interpret a schema-valid combination.

Example:

```
JSON
{
  "status": ["upcoming", "unconfirmed"],
  "afterStart": "2030-01-01T00:00:00Z",
  "beforeEnd": "2030-02-01T00:00:00Z",
  "sortStart": "asc",
  "take": 50,
  "skip": 0
}
```

Read by booking or seat uid

Read a regular or recurring booking set with:

```
JSON
{
  "booking_uid": "booking-provider-uid"
}
```

Read a seated booking with:

```
JSON
{
  "seat_uid": "seat-provider-uid"
}
```

Both `booking_uid` and `seat_uid` must be non-empty strings. Treat them as sensitive external handles. The connector percent-encodes them into provider paths, so keep raw values out of application logs, user-visible URLs, and metric labels.

The three read capabilities are synchronous, low risk, retry-safe, and eligible for connector retry. Their idempotency key is optional and action-scoped. They still require authenticated account routing and restricted-data policy.

Create a booking

`cap:cal_diy:booking.create` requires `start`, `attendee`, and exactly one event-type selector. A minimal id-based input is:

```
JSON
{
  "start": "2030-01-03T10:00:00Z",
  "eventTypeId": 42,
  "attendee": {
    "name": "Avery Patel",
    "email": "avery@example.com",
    "timeZone": "Europe/London"
  }
}
```

Event-type selectors

Choose exactly one branch:

Branch	Required fields	Optional related field
Provider id	`eventTypeId`	None
User-owned slug	`eventTypeSlug`, `username`	`organizationSlug`
Team-owned slug	`eventTypeSlug`, `teamSlug`	`organizationSlug`

The id is an integer. Slugs and usernames must be non-empty strings. Satisfying more than one branch makes the `oneOf` input invalid. Do not mix selector fields even when an incomplete second branch could still pass the schema.

Attendee

`attendee` requires a non-empty `name`, a non-empty `timeZone`, and at least one of `email` or `phoneNumber`:

Field	Constraint
`name`	Non-empty string
`timeZone`	Non-empty string
`email`	Email string; required when phone is absent
`phoneNumber`	+ plus 7 through 15 digits; required when email is absent
`language`	The 43-value Cal.diy locale enum

The locale enum is the same source-owned set listed in the profile capability reference. Additional attendee fields are rejected.

Optional booking fields

Field	Constraint or role
`bookingFieldsResponses`	Object with provider-defined response keys and values
`guests`	Unique array of email strings
`meetingUrl`	URI string; deprecated
`location`	One supported create-location variant
`metadata`	At most 46 application entries; string values only
`lengthInMinutes`	Integer at least `1`
`routing`	Routing-form response and team-member selection
`emailVerificationCode`	String
`recurrenceCount`	Integer at least `1`

`metadata` property names have at most 40 characters and values have at most 500 characters. The connector reserves and writes these keys before provider dispatch:

- `aip_action_id`;
- `aip_operation_id`;
- `aip_idempotency_hash`;
- `aip_transaction_id` when a transaction id exists.

The idempotency value is hashed before insertion. Do not supply those reserved keys as application metadata; connector values overwrite matching keys.

`routing` requires integer `responseId` and an array of integer `teamMemberIds`. It may also contain `teamMemberEmail`, `skipContactOwner`, `crmAppSlug`, and `crmOwnerRecordType`. Additional routing fields are rejected.

Create locations

Create accepts these object variants:

Type	Required value
`integration`	One of the 29 supported integration names
`attendeeAddress`	Non-empty `address`
`attendeePhone`	E.164-like `phone`
`attendeeDefined`	Non-empty `location`
`address`, `link`, `phone`, `organizersDefaultApp`	Type discriminator only

The integration enum is listed in the event-type capability reference. A non-empty free-form string is also accepted for compatibility but is marked deprecated. Do not introduce new use of the deprecated string or `meetingUrl`.

Plan booking creation

Booking create cannot use the direct invocation path. The connector returns `connector.cal_diy.transaction_required` until the action follows the AIP plan and commit lifecycle.

The plan:

1. validates the exact create input;
2. queries `slot.list` from the requested start through one day later;

3. requests `time` format;
4. verifies that the response contains the exact requested date-time start;
5. returns a completed plan result without committing a provider side effect.

The plan result identifies the operation, API version, provider path, input hash, validation class `downstream_slot_availability`, and `side_effect_committed: false`.

A passing plan is an observation, not a reservation. Availability can change before commit.

Reviewed planner limitation

At the reviewed source revision, the slot probe forwards `eventTypeId`, `eventTypeSlug`, `username`, `organizationSlug`, `timeZone`, and a field named `duration` when present. The create schema exposes `teamSlug`, nested `attendee.timeZone`, and `lengthInMinutes`; it does not accept top-level `timeZone` or `duration`.

Consequently, the planner does not forward a team selector's `teamSlug` or a custom `lengthInMinutes` or attendee time zone into the provider availability query. It checks only whether the requested start appears in the returned slots. Treat team-slug, custom-length, and time-zone plan coverage as an implementation limitation at this revision; do not describe it as full downstream validation.

Reschedule a booking

Reschedule requires a booking uid and new start:

```
JSON
{
  "booking_uid": "booking-provider-uid",
  "start": "2030-01-03T11:00:00Z",
  "rescheduledBy": "operator@example.com",
  "reschedulingReason": "Customer requested a later time"
}
```

Optional fields are `rescheduledBy` as an email, `reschedulingReason` as a string, non-empty `seatUid`, and `emailVerificationCode` as a string.

For a seat-specific reschedule, include `seatUid` and omit `reschedulingReason`. The schema rejects an input containing both. A regular reschedule omits `seatUid` and may include the reason.

Reschedule is high risk and requires plan before commit. Its plan reads the booking, requires integer `/data/eventTypeId` in the provider result, queries availability for that event type while passing the booking uid as `bookingUidToReschedule`, and checks the exact requested start. The plan's validation class is `booking_state_and_downstream_slot_availability`.

If the booking result omits `data.eventTypeId`, planning fails before the reschedule request. A passing plan still does not reserve the new slot.

Cancel a booking

Cancel requires the booking uid:

```
JSON
{
  "booking_uid": "booking-provider-uid",
  "cancellationReason": "Customer cancelled",
  "cancelSubsequentBookings": true
}
```

For a seat-specific cancellation, include non-empty `seatUid` and omit `cancelSubsequentBookings`. The schema rejects both fields together. `cancellationReason` remains optional in either branch.

Cancel is a high-risk destructive mutation and requires plan before commit. Its plan reads the target booking and returns validation class `downstream_target_exists`. It does not simulate cancellation or prove that notifications will be delivered.

Cancellation has no rollback. It is the named best-effort compensation for a booking create, but executing it cannot erase already delivered notifications, calendar observations, or other downstream effects.

Confirm or decline a booking

Confirm accepts only:

```
JSON
{
  "booking_uid": "booking-provider-uid"
}
```

Decline optionally adds a reason:

```
JSON
{
  "booking_uid": "booking-provider-uid",
  "reason": "Requested time cannot be supported"
}
```

Both are high risk. Confirm is not marked destructive; decline is. Neither requires a plan before commit, but both require approval and first read the target booking. Both carry the `send_message` side-effect marker.

Mark absence

Mark host absence, attendee absence, or both:

```
JSON
{
  "booking_uid": "booking-provider-uid",
  "host": false,
  "attendees": [
    {
      "email": "avery@example.com",
      "absent": true
    }
  ]
}
```

`attendees` must be non-empty when present. Each item requires an email and a Boolean `absent`; additional item fields are rejected.

Only `booking_uid` is required by the schema, so an input with no `host` or `attendees` is schema-valid. Do not use an empty absence update as a probe; Cal.diy may reject it or apply provider-specific behavior.

This high-risk mutation first reads the booking. It does not carry the connector's `send_message` marker.

Reassign a booking

Automatic round-robin reassignment accepts only the booking uid:

```
JSON
{
  "booking_uid": "booking-provider-uid"
}
```

Reassignment to a selected user also requires an integer `user_id` and may include a reason:

```
JSON
{
  "booking_uid": "booking-provider-uid",
  "user_id": 73,
  "reason": "Route to the account owner"
}
```

The connector puts both identifiers in the provider path and sends `reason` in the JSON body. Both operations are high risk, first read the booking, and do not carry the connector's `send_message` marker. Provider behavior may still affect calendars or notifications; the absence of that marker is not a delivery guarantee.

Update the booking location

Location update requires only the booking uid; include a location to make an intentional change:

```
JSON
{
  "booking_uid": "booking-provider-uid",
  "location": {
    "type": "link",
    "link": "https://meet.example.com/room-42"
  }
}
```

Update accepts:

- `integration` with one supported integration name;
- `attendeeAddress` with non-empty `address`;
- `attendeePhone` with E.164-like `phone`;
- `attendeeDefined` with non-empty `location`;
- `address`, `link`, or `phone` with the corresponding non-empty value.

Unlike create, location update does not accept `organizersDefaultApp`, a bare deprecated string, or discriminator-only `address`, `link`, or `phone`.

The schema permits omission of `location`. Do not send that empty update as a probe. This medium-risk mutation first reads the booking and does not carry the `send_message` marker.

Preflight and planning matrix

Mutation	Read or validation before provider mutation	Plan required
Create	Exact-start slot query	Yes
Reschedule	Booking read, event-type extraction, exact-start slot query	Yes
Cancel	Booking read	Yes
Confirm	Booking read	No
Decline	Booking read	No
Mark absent	Booking read	No
Reassign	Booking read	No
Reassign to user	Booking read	No
Location update	Booking read	No

A preflight detects some stale or missing targets before mutation. It does not reserve state, authorize the action, guarantee provider acceptance, or make a write safe to retry.

Shared mutation contract

The nine mutations require:

- approval selected by tenant policy;
- a 900,000 ms approval validity window;

- reason, input snapshot, and policy-decision evidence;
- an external-account-scoped idempotency key;
- input-hash revalidation on key collision.

All mutations are synchronous, advertise no connector retry, and are retry unsafe. Preserve the same key only for the same capability, account, and canonical input. Reconcile uncertain provider dispatch instead of creating a replacement key.

Create, reschedule, cancel, confirm, and decline carry `send_message` in addition to write and external-network side effects. This marker means a provider operation may affect customer-visible notifications or connected calendar state; it does not prove delivery.

All mutations advertise `dry_run`, `plan`, `commit`, and `reconcile`. Create, reschedule, and cancel require a validated plan and have downstream-validation dry-run fidelity. The remaining mutations have policy-and-schema fidelity and do not require plan before commit.

Booking create names `booking.cancel` as best-effort compensation. The contract window is 24 hours and compensation requires approval. It is a new governed action, not an automatic or atomic rollback. Every other operation on this page has no compensating capability.

Result and data contract

The connector returns the validated provider JSON body as AIP output. It requires a non-empty string `status`; `data`, `pagination`, and additional provider fields are optional. The connector does not publish an exhaustive booking response schema.

Before returning output, the connector recursively redacts fields whose normalized names identify secrets, API keys, or tokens. This targeted redaction does not declassify booking content. Inputs, outputs, approval snapshots, receipts, and diagnostic evidence remain restricted and PII-bearing.

Failures and recovery

Failure	Relevant cause	Safe response
`connector.cal_diy.transaction_required`	Create, reschedule, or cancel used direct invocation	Start the governed plan-and-commit lifecycle
`connector.cal_diy.slot_unavailable`	Planned create or reschedule start is absent from current slots	Choose a new start and produce a new plan
`connector.cal_diy.invalid_action`	Schema mismatch or reschedule booking result omitted `eventTypeId`	Correct input or investigate provider response drift
`connector.cal_diy.policy`	Mutation omitted its idempotency key	Preserve approved input and supply the required key
`connector.cal_diy.idempotency_collision`	One key owns different canonical input	Stop and inspect the original action
`connector.cal_diy.in_flight`	The matching mutation is executing	Read durable state and wait
`connector.cal_diy.outcome_unknown`	Prior provider dispatch cannot be proved	Reconcile; do not repeat the mutation
`connector.cal_diy.authentication`	Provider returned `401` or `403`	Verify account binding, credential revision, and booking ownership
`connector.cal_diy.state_conflict`	Provider returned `409` or `412`	Read current booking state and reconsider the transition
`connector.cal_diy.rate_limited`	Provider returned `429`	Retry 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`	Provider success lacked a valid `status`	Preserve evidence and reconcile a mutation
`connector.cal_diy.reconciliation_not_found`	Durable provider operation cannot be found	Stop and escalate with the original action and operation ids

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 a booking transition

For every operation, retain the capability id, tenant and external account, AIP action id, input hash, provider request id when present, and completed or reconciled durable state.

For create, reschedule, or cancel, additionally verify the plan id and the exact input bound to it. For create, confirm the provider booking uid before using it in later actions. For state transitions, read the booking after a completed result and compare only the intended fields.

Do not use a follow-up read to justify replaying an uncertain mutation. Resolve the original provider operation through reconciliation and retain the evidence strength of that result.

These checks validate application handling. They do not establish live Cal.diy qualification or production readiness for a specific deployment.

Related documentation

- Cal.diy capability index
- Slots and reservations
- Booking attendees and artifacts
- Approvals and policy
- Transactions and compensation