Cal.diy calendars and free/busy

Use these capabilities to inspect connected calendars and availability, work with events, choose the destination for new bookings, or change conflict detection. This page owns all 21 operations in the calendar family.

SECTION	Connectors
TYPE	Connector
PROTOCOL	AIP 1.0
SOURCE	docs/connectors/cal-diy/capabilities/calendars-and-free-busy.md

All operations use provider API version `2024-08-13`. Every input and result in this family is restricted, contains PII, and requires redaction.

Choose a calendar surface

Cal.diy exposes two event surfaces. Choose one before constructing input:

Surface	Selector	Provider scope
Unified connection	Integer `connection_id`; optional `calendarId`	A connection returned by Cal.diy
Direct provider	`calendar` fixed to `google`	Google event and free/busy routes only

The broader provider checks accept `google`, `office365`, or `apple`. That does not expand the direct event surface: its published schema still requires `"calendar": "google"`.

Choose an operation

Connection and availability reads

Capability	Provider request	Input
`cap:cal_diy:calendar.list`	`GET /v2/calendars`	Empty object
`cap:cal_diy:calendar.provider.check`	`GET /v2/calendars/{calendar}/check`	Provider
`cap:cal_diy:calendar.ics_feed.check`	`GET /v2/calendars/ics-feed/check`	Empty object
`cap:cal_diy:calendar.busy_time.list`	`GET /v2/calendars/busy-times`	Range and calendars
`cap:cal_diy:calendar.connection.list`	`GET /v2/calendars/connections`	Empty object
`cap:cal_diy:calendar.connection.free_busy.get`	`GET /v2/calendars/connections/{connection_id}/freebusy`	Connection and range
`cap:cal_diy:calendar.free_busy.get`	`GET /v2/calendars/{calendar}/freebusy`	Google and range

All seven operations are low-risk, approval-free reads at the capability contract level.

Unified connection events

Capability	Provider request	Risk	Approval
`cap:cal_diy:calendar.connection.event.list`	`GET /v2/calendars/connections/{connection_id}/events`	Low	No
`cap:cal_diy:calendar.connection.event.get`	`GET /v2/calendars/connections/{connection_id}/events/{event_id}`	Low	No
`cap:cal_diy:calendar.connection.event.create`	`POST /v2/calendars/connections/{connection_id}/events`	Medium	Required
`cap:cal_diy:calendar.connection.event.update`	`PATCH /v2/calendars/connections/{connection_id}/events/{event_id}`	Medium	Required
`cap:cal_diy:calendar.connection.event.delete`	`DELETE /v2/calendars/connections/{connection_id}/events/{event_id}`	High	Required

Direct Google events

Capability	Provider request	Risk	Approval
`cap:cal_diy:calendar.event.list`	`GET /v2/calendars/{calendar}/events`	Low	No
`cap:cal_diy:calendar.event.get`	`GET /v2/calendars/{calendar}/events/{event_uid}`	Low	No
`cap:cal_diy:calendar.event.create`	`POST /v2/calendars/{calendar}/events`	Medium	Required
`cap:cal_diy:calendar.event.update`	`PATCH /v2/calendars/{calendar}/events/{event_uid}`	Medium	Required
`cap:cal_diy:calendar.event.delete`	`DELETE /v2/calendars/{calendar}/events/{event_uid}`	High	Required

For every direct operation in this table, `calendar` must equal `google`.

Routing and conflict detection

Capability	Provider request	Risk	Approval
`cap:cal_diy:calendar.disconnect`	`POST /v2/calendars/{calendar}/disconnect`	High	Required
`cap:cal_diy:calendar.destination.update`	`PUT /v2/destination-calendars`	Medium	Required
`cap:cal_diy:calendar.selected.add`	`POST /v2/selected-calendars`	Medium	Required
`cap:cal_diy:calendar.selected.delete`	`DELETE /v2/selected-calendars`	High	Required

Disconnect, both event deletes, and selected-calendar delete are destructive.

Inspect calendar connections

List connected calendars with an empty input:

```
JSON
{}
```

Use `calendar.provider.check` to test one provider connection:

```
JSON
{
  "calendar": "google"
}
```

The exact provider enum is `google`, `office365`, or `apple`. Provider check shows whether the authenticated integration is connected and healthy at that request boundary. It does not prove that every calendar is visible or writable.

`calendar.ics_feed.check` also accepts an empty object and checks the authenticated user's ICS feed connection. `calendar.connection.list` returns credential-safe unified connection identifiers; the connector does not publish their provider-owned response fields.

These reads inspect connections that already exist. Provider connect and save routes that establish browser OAuth sessions or credentials remain outside the agent-callable business-operation surface.

Read busy time across selected calendars

`calendar.busy_time.list` requires a time zone, a bounded range, and at least one explicit calendar:

```
JSON
{
  "timeZone": "UTC",
  "dateFrom": "2026-08-03",
  "dateTo": "2026-08-04T18:00:00Z",
  "calendarsToLoad": [
    {
      "credentialId": 42,
      "externalId": "primary@example.com"
    }
  ]
}
```

Field	Constraint
`timeZone`	Non-empty string
`dateFrom`, `dateTo`	ISO date or date-time string
`calendarsToLoad`	Array with at least one object
`credentialId`	Integer required in each object
`externalId`	Non-empty string required in each object

The client sends these values as query parameters. Nested calendar objects use bracket notation such as `calendarsToLoad[0][credentialId]`. The schema does not set a maximum or require unique calendar objects, and it does not assert that `dateFrom` precedes `dateTo`.

Busy intervals are scheduling evidence, not a reservation or a guarantee that a later booking will succeed.

Read unified connection events

List events for a unified connection with a required range:

```
JSON
{
  "connection_id": 73,
  "from": "2026-08-03T00:00:00Z",
  "to": "2026-08-04T00:00:00Z",
  "timeZone": "UTC",
  "calendarId": "team@example.com"
}
```

`connection_id`, `from`, and `to` are required. `from` and `to` accept either an ISO date or date-time. Optional `timeZone` and `calendarId` must be non-empty. The connection id is a path segment; all range fields and `calendarId` become query parameters.

Read one event with `connection_id` and a non-empty `event_id`. An optional `calendarId` selects a calendar within that connection and is sent as a query parameter:

```
JSON
{
```

```
  "connection_id": 73,
  "event_id": "provider-event-uid",
  "calendarId": "team@example.com"
}
```

Create a unified connection event

Create requires the connection, title, start, and end:

```
JSON
{
  "connection_id": 73,
  "calendarId": "team@example.com",
  "title": "Architecture review",
  "start": {
    "time": "2026-08-03T10:00:00Z",
    "timeZone": "UTC"
  },
  "end": {
    "time": "2026-08-03T10:30:00Z",
    "timeZone": "UTC"
  },
  "description": "Review the proposed integration boundary.",
  "attendees": [
    {
      "email": "avery@example.com",
      "name": "Avery Patel"
    }
  ]
}
```

`title` must be non-empty. Both `start` and `end` require a date-time `time` and a non-empty `timeZone`. `description` may be a string or `null`. `attendees` may contain objects with a required `email` and optional `name`; the array has no published minimum or uniqueness constraint. Additional fields are rejected at the event, date-time, and attendee object boundaries.

`calendarId` is optional. The client puts it in the query while sending event details as JSON. The connector contract carries `write` and `external_network`, but not `send_message`. Attendee input is therefore not evidence that the provider sent an invitation.

The create operation names `calendar.connection.event.delete` as best-effort compensation. Compensation requires a separate approval and is not automatic.

Update or delete a unified connection event

Update requires `connection_id` and `event_id`. It may include `calendarId` and any of these body fields:

Field	Constraint
`title`	String; an empty string passes the connector schema
`start`, `end`	Object with optional date-time `time` and string `timeZone`
`description`	String or `null`
`attendees`	Array or `null`
`status`	`accepted`, `pending`, `declined`, `cancelled`, or `null`

An attendee update requires `email` and may include `name`, `self`, `optional`, `host`, and `responseStatus`. Response status accepts `accepted`, `pending`, `declined`, `needsAction`, or `null`.

The schema allows an identifier-only update and empty `start` or `end` objects. Such input can pass local validation without expressing a useful provider change. Include only an intentional, provider-supported patch.

Delete uses `connection_id` and `event_id`, plus optional `calendarId`. The two ids become path segments and `calendarId` becomes a query parameter. Delete is high risk and has no rollback contract.

Both update and delete first read the exact unified event. A successful preflight proves only that the event was readable before mutation.

Read, create, or change a direct Google event

The direct surface mirrors the unified event model with these selector changes:

- every input requires `"calendar": "google"`;
- list also requires `from` and `to`, with optional `timeZone` and `calendarId`;
- get and delete require a non-empty `event_uid`;
- create has no `calendarId` field;
- update requires `event_uid` and uses the shared optional patch fields.

Create and update use the field constraints described for unified events. Direct create names `calendar.event.delete` as best-effort compensation. Direct update and delete first run `calendar.event.get` for the same `calendar` and `event_uid`.

The direct schema does not publish Office 365 or Apple event variants. Use a unified connection when its returned identifier is the intended routing model.

Read free/busy

Unified free/busy requires `connection_id`, `from`, and `to`. Direct free/busy requires `calendar` fixed to `google`, `from`, and `to`. Both accept optional non-empty `timeZone`; all range fields are query parameters.

```
JSON
{
  "connection_id": 73,
  "from": "2026-08-03",
  "to": "2026-08-04",
  "timeZone": "UTC"
}
```

The input schema accepts dates or date-times and does not enforce ordering or a maximum interval. A free/busy response is provider-owned and restricted. It does not by itself apply schedule rules, event-type buffers, reservations, or booking policy.

Change the destination calendar

`calendar.destination.update` changes where new bookings are written:

```
JSON
{
  "integration": "google_calendar",
  "externalId": "primary@example.com"
}
```

`integration` must be `apple_calendar`, `google_calendar`, or `office365_calendar`. `externalId` is required and non-empty. An optional non-empty `delegationCredentialId` selects delegated provider credentials.

The connector first lists calendars. This does not prove that the requested `externalId` appeared in that result; provider validation still decides the mutation. The operation is medium risk and has no compensating capability.

Add or remove a conflict calendar

Add requires an integration name, external calendar id, and integer credential id:

```
JSON
{
```

```
  "integration": "google_calendar",
  "externalId": "team@example.com",
  "credentialId": 42
}
```

All three fields are required. `integration` and `externalId` must be non-empty; `credentialId` is an integer. Optional `delegationCredentialId` must be non-empty. The schema does not constrain `integration` to the destination calendar enum.

Remove uses the same route but requires `credentialId` as a non-empty string, not an integer. The client sends `integration`, `externalId`, `credentialId`, and optional `delegationCredentialId` as query parameters on `DELETE`.

Both operations first list calendars. Add declares `calendar.selected.delete` as best-effort compensation. Remove is destructive and has no further rollback.

Disconnect a provider credential

Disconnect requires a provider and integer credential id:

```
JSON
{
  "calendar": "office365",
  "id": 42
}
```

`calendar` must be `google`, `office365`, or `apple`. The client sends a `POST` to the provider-specific disconnect path and leaves `id` in the JSON body. Before dispatch, it checks that provider connection without using the id.

Integer identifiers in this family have no additional minimum in the published schema. Cal.diy can still reject a value outside the provider's domain.

The operation is high risk and destructive. It deletes an owned credential and invalidates the connection. No compensation or automatic credential recovery is declared.

Preflight reads

Mutation	Connector preflight
Unified or direct event create	`calendar.list`
Destination update	`calendar.list`
Selected-calendar add or delete	`calendar.list`
Unified event update or delete	Exact unified event get
Direct event update or delete	Exact direct event get
Calendar disconnect	Provider check for `calendar`

A preflight is a point-in-time read, not a lock, plan, reservation, or approval.

Shared read contract

All 11 reads are low risk, synchronous, retry-safe, and eligible for connector retry. Their idempotency key is optional, action-scoped, and collision-checked against the input hash. No capability-level approval is required.

Identity, tenant, external-account, credential-scope, and restricted-data policy still apply to every read.

Shared mutation contract

All ten mutations require:

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

Six non-destructive mutations are medium risk. Calendar disconnect, both event deletes, and selected-calendar delete are high risk and destructive. None of the ten mutations carries the `send_message` side-effect marker.

Event create on either surface and selected-calendar add each declare one best-effort delete compensation. All other mutations declare rollback as not supported. Never interpret a compensation declaration as automatic execution or restoration of invitations, notifications, external state, or history.

Result and redaction contract

The connector returns the validated provider JSON body as AIP output. A non-empty string `status` is required. `data`, `pagination`, and additional provider fields are optional; their shapes are not guaranteed by the connector schema.

The connector recursively replaces secret-, key-, and token-named values with `[REDACTED]`. This targeted redaction does not remove event titles, attendee addresses, calendar ids, busy intervals, descriptions, or other PII unless a field name matches the secret redactor. Keep the complete output restricted.

Failures and recovery

Failure	Relevant cause	Safe response
`connector.cal_diy.invalid_action`	Missing selector, invalid range shape, invalid enum, or unknown field	Correct the exact input before another call
`connector.cal_diy.policy`	Mutation omitted its required idempotency key	Preserve approved input and add the missing key
`connector.cal_diy.idempotency_collision`	One key owns different mutation input	Stop and inspect the original action
`connector.cal_diy.in_flight`	A matching calendar mutation is executing	Read durable state and wait
`connector.cal_diy.outcome_unknown`	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, credential revision, and calendar ownership
`connector.cal_diy.state_conflict`	Provider returned `409` or `412`	Refresh event or calendar state and review the mutation
`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`	Successful response lacked valid `status`	Preserve evidence and reconcile a mutation

Permanent 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 calendar handling

For a read, retain the capability, exact selector and range, actor, tenant, external account, provider status, and provider request id when present. Verify that the result remains inside its approved restricted-data destination.

For a mutation, also retain approval evidence and the original idempotency key. Wait for a completed durable result, then read the exact event or list calendar state. For an uncertain dispatch, reconcile the original operation before any new action.

These checks validate application handling. They do not prove invitation delivery, provider-wide visibility, live availability, cross-provider qualification, or production readiness.

Related documentation

- Cal.diy capability index
- Core booking capabilities
- Authentication and tenant routing
- Transactions and compensation
- Error reference