Diagnose Cal.diy availability and calendar conflicts

Use this guide when an expected Cal.diy slot is missing or when you need to change the schedule or calendars that can contribute conflicts. You will capture one repeatable baseline, change at most one input layer, and compare the same obser

SECTION	Connectors
TYPE	Connector
PROTOCOL	AIP 1.0
SOURCE	docs/connectors/cal-diy/guides/availability-and-calendar-conflicts.md

The connector exposes the relevant reads and mutations, but it does not expose an operation that explains why Cal.diy returned or omitted a slot. Treat the result as evidence at one time, not as a causal diagnosis or a capacity hold.

Keep the six layers separate

Availability is not one connector object. Record which layer each observation or change belongs to:

Layer	What it can show or change	What it does not prove
Event type	`scheduleId`, booking window, notice, duration, and buffers in provider-owned state	That the linked schedule or every rule is valid
Schedule	Weekly intervals and dated overrides owned by one schedule	That the event type uses this schedule
Conflict selection	Which external calendars Cal.diy should consider	The events or busy intervals on those calendars
Busy/free-busy	Explicit intervals returned for an exact account, range, and calendar set	That Cal.diy used the same set to calculate slots
Slots	Provider availability for one selector, range, and query shape	A reservation or the reason for a missing time
Booking plan	Exact-start validation for one booking intent	General availability or a capacity hold

Do not substitute one layer for another. In particular, an empty busy result is not a slot result, and a slot result does not reveal the selected conflict calendar configuration.

Prerequisites

You need:

- an admitted and healthy Cal.diy connector for the intended tenant;
- a native AIP endpoint and token already bound to that tenant;
- the integer id of one event type and one trusted schedule id;
- the external calendar id, integration name, and credential id when inspecting

busy time or changing conflict selection;

- a trusted tenant-policy approval path for mutations;

- durable storage for action ids, mutation idempotency keys, results, and

provider request identifiers.

Set the client coordinates without putting a provider credential in the shell:

```
SH
export AIP_URL='https://aip.example.com'
export AIP_TOKEN_FILE='/run/secrets/aip-native-token'
```

The authenticated deployment context selects the tenant, external account, connector route, and provider credential. Do not add those routing values to the Cal.diy action input.

1. Confirm the admitted contracts

Inspect the exact capabilities before using them:

```
SH
for capability in \
  cap:cal_diy:event_type.get \
  cap:cal_diy:schedule.get \
  cap:cal_diy:schedule.update \
  cap:cal_diy:calendar.list \
  cap:cal_diy:calendar.busy_time.list \
  cap:cal_diy:calendar.selected.add \
  cap:cal_diy:calendar.selected.delete \
  cap:cal_diy:slot.list
do
  aipctl \
    --native-bearer-token-file "$AIP_TOKEN_FILE" \
    capability list "$AIP_URL" \
    --capability-id "$capability" \
    --limit 1
done
```

Record the catalog revision and schema and contract digests. Confirm that the reads are low risk and approval-free. `schedule.update` and `calendar.selected.add` are medium risk; `calendar.selected.delete` is high risk and destructive. All three mutations require tenant-policy approval and an external-account-scoped idempotency key.

These mutations do not require a downstream plan before commit. Their contract advertises policy-and-schema dry-run fidelity. A transaction plan may perform the operation's preflight read, but it does not calculate future slots or lock provider state.

2. Freeze one diagnostic scope

Write down one scope before making any call:

Coordinate	Example	Keep fixed because
Event type	`42`	A different selector can use different rules
Schedule	`41`	Schedule reads do not prove event-type linkage
Range	`2030-01-03` to `2030-01-04`	A changed interval produces a different observation
Time zone	`Europe/London`	Local-time interpretation can change returned times
Calendar	Credential `42`, `team@example.com`	Busy reads require an explicit calendar set
Slot format	`time`	A changed format can change result representation

Use the same tenant and external account for the full investigation. If any coordinate changes, start a new comparison instead of treating the result as the after-state of this one.

3. Read the event type and schedule

Write `event-type-get.json`:

```
JSON
{
  "event_type_id": 42
}
```

Read the event type:

```
SH
export EVENT_TYPE_ACTION_ID='act_cal_availability_event_type_001'

aipctl \
  --native-bearer-token-file "$AIP_TOKEN_FILE" \
  action call "$AIP_URL" \
  cap:cal_diy:event_type.get \
  --action-id "$EVENT_TYPE_ACTION_ID" \
  --mode sync \
  --input @event-type-get.json
```

Inspect the actual provider output for timing controls such as `scheduleId`, `minimumBookingNotice`, `beforeEventBuffer`, `afterEventBuffer`, `lengthInMinutes`, and booking-window values. The connector's common output schema does not guarantee those fields, so absence is not a default value.

Write `schedule-get.json`:

```
JSON
{
  "schedule_id": 41
}
```

Read the schedule selected for this investigation:

```
SH
export SCHEDULE_ACTION_ID='act_cal_availability_schedule_001'

aipctl \
  --native-bearer-token-file "$AIP_TOKEN_FILE" \
  action call "$AIP_URL" \
  cap:cal_diy:schedule.get \
  --action-id "$SCHEDULE_ACTION_ID" \
  --mode sync \
  --input @schedule-get.json
```

Record its time zone, weekly availability, dated overrides, and provider identifier when present. Compare the returned id with the event type's actual `scheduleId` value. Do not assume that `SCHEDULE_ID` is linked merely because the schedule exists.

The schedule schema checks field shapes, but not interval ordering, overlap, time-zone validity, or daylight-saving behavior.

4. Capture the baseline slot result

Write `slot-baseline.json` with one selector and the frozen range:

```
JSON
{
  "start": "2030-01-03",
  "end": "2030-01-04",
  "eventTypeId": 42,
  "timeZone": "Europe/London",
  "format": "time"
}
```

Run the baseline read and retain its complete result and observation time:

```
SH
export BASELINE_SLOT_ACTION_ID='act_cal_availability_slots_before_001'
```

```
aipctl \
  --native-bearer-token-file "$AIP_TOKEN_FILE" \
  action call "$AIP_URL" \
  cap:cal_diy:slot.list \
  --action-id "$BASELINE_SLOT_ACTION_ID" \
  --mode sync \
  --input @slot-baseline.json
```

The slot operation checks that `start` and `end` have ISO date or date-time shape, but it does not prove chronological order or validate the time-zone name. A completed result reports Cal.diy's response at that request boundary. It does not reserve a returned time.

5. Inspect explicit busy intervals

First list the calendars visible through the current account route. Save an empty object as `calendar-list.json`:

```
JSON
{}
```

Run the calendar read:

```
SH
export CALENDAR_LIST_ACTION_ID='act_cal_availability_calendars_001'

aipctl \
  --native-bearer-token-file "$AIP_TOKEN_FILE" \
  action call "$AIP_URL" \
  cap:cal_diy:calendar.list \
  --action-id "$CALENDAR_LIST_ACTION_ID" \
  --mode sync \
  --input @calendar-list.json
```

Validate the intended credential and external calendar against the actual provider response when those fields are present. The common connector output schema does not guarantee their shape, and this operation is not a dedicated selected-conflict-calendar read.

Write `busy-time.json`:

```
JSON
{
  "timeZone": "Europe/London",
  "dateFrom": "2030-01-03",
  "dateTo": "2030-01-04",
  "calendarsToLoad": [
    {
      "credentialId": 42,
      "externalId": "team@example.com"
    }
  ]
}
```

Read busy time for exactly that calendar set:

```
SH
export BUSY_ACTION_ID='act_cal_availability_busy_001'

aipctl \
  --native-bearer-token-file "$AIP_TOKEN_FILE" \
  action call "$AIP_URL" \
  cap:cal_diy:calendar.busy_time.list \
  --action-id "$BUSY_ACTION_ID" \
  --mode sync \
  --input @busy-time.json
```

Each `calendarsToLoad` object requires an integer `credentialId` and non-empty `externalId`. The connector encodes the array as nested query parameters. It does not infer the selected conflict-calendar set for you.

Calendar reads are restricted, PII-bearing data. Store event details, external ids, busy intervals, and results under the deployment's restricted-data policy.

6. Choose one change, if any

Use the evidence to choose one next action:

Observation	Next check or change	Do not conclude
Event type points to another schedule	Read that exact schedule before changing anything	The schedule you first read controls this event type
Weekly interval or dated override excludes the time	Review one schedule patch	That a patch alone will create a slot
Notice, buffer, duration, or booking window excludes the time	Review the event-type capability reference	That the calendar is the cause
Explicit busy interval overlaps the expected time	Confirm the calendar identity and provider event	That it is selected for conflict detection
Calendar should contribute conflicts but is not configured	Review one selected-calendar add	That `calendar.list` proves selection state
All inspected inputs appear compatible	Preserve evidence and escalate provider-side diagnosis	That a missing connector explanation is proof of no conflict

Change at most one layer before repeating the reads. This makes the comparison useful, but it still cannot establish provider causality.

7. Update one schedule field safely

Skip this step if the schedule is already correct. Use an id returned by a trusted read and include an intentional patch field. For example, write `schedule-update.json`:

```
JSON
{
  "schedule_id": 41,
  "availability": [
    {
      "days": ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"],
      "startTime": "09:00",
      "endTime": "17:00"
    }
  ]
}
```

Review the complete desired array before approval. The connector does not define whether Cal.diy merges or replaces availability arrays.

Use one stable action id and idempotency key:

```
SH
export SCHEDULE_UPDATE_ACTION_ID='act_cal_availability_schedule_update_001'
export SCHEDULE_UPDATE_KEY='cal-availability-schedule-update-001'

aipctl \
  --native-bearer-token-file "$AIP_TOKEN_FILE" \
  action call "$AIP_URL" \
  cap:cal_diy:schedule.update \
  --action-id "$SCHEDULE_UPDATE_ACTION_ID" \
  --idempotency-key "$SCHEDULE_UPDATE_KEY" \
  --input @schedule-update.json
```

The connector performs an exact schedule read before dispatch. That preflight is not a lock and does not validate the new intervals. A trusted approver must review the input snapshot and policy evidence. Approval resumes the same action; do not submit a replacement action while it is waiting.

`schedule.update` accepts a string or integer id, although schedule get and delete accept only an integer. Use the integer returned by the trusted read. Update has no compensating capability.

8. Add one conflict calendar safely

Skip this step if conflict selection is not the intended change. Write `conflict-calendar-add.json`:

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

Use the exact external id and integer credential id reviewed in the current account route:

```
SH
export CALENDAR_ADD_ACTION_ID='act_cal_availability_calendar_add_001'
export CALENDAR_ADD_KEY='cal-availability-calendar-add-001'

aipctl \
  --native-bearer-token-file "$AIP_TOKEN_FILE" \
  action call "$AIP_URL" \
  cap:cal_diy:calendar.selected.add \
  --action-id "$CALENDAR_ADD_ACTION_ID" \
  --idempotency-key "$CALENDAR_ADD_KEY" \
  --input @conflict-calendar-add.json
```

The connector lists calendars before dispatch, but it does not verify that the requested `externalId` appeared in that result. Provider validation still owns the mutation decision.

The add contract names `calendar.selected.delete` as best-effort compensation within 24 hours. Compensation is a separate, approval-required destructive action. The declaration does not transform add input into valid delete input; delete requires the credential id as a string.

9. Wait for the original mutation

For the one mutation you submitted, read the same action until it reaches a terminal state. For a schedule update:

```
SH
aipctl \
  --native-bearer-token-file "$AIP_TOKEN_FILE" \
  action status "$AIP_URL" \
  "$SCHEDULE_UPDATE_ACTION_ID" \
  --include-result \
  --include-receipts \
  --wait-ms 30000
```

Use `CALENDAR_ADD_ACTION_ID` instead if you added a conflict calendar. Preserve the approval evidence, original input, idempotency key, provider request id, and provider operation id when present.

Only a terminal `completed` result proves that the connector accepted the provider response and durably settled the action. It does not prove immediate availability propagation.

10. Repeat the same observations

After a completed mutation:

1. read the same event type again;
2. read the same schedule again after a schedule change;
3. list calendars again after a selected-calendar change;
4. repeat `busy-time.json` if busy evidence remains relevant;
5. submit the unchanged `slot-baseline.json` under a new action id.

Repeat the slot query without editing its selector, range, time zone, or format:

```
SH
export AFTER_SLOT_ACTION_ID='act_cal_availability_slots_after_001'

aipctl \
  --native-bearer-token-file "$AIP_TOKEN_FILE" \
  action call "$AIP_URL" \
  cap:cal_diy:slot.list \
```

```
  --action-id "$AFTER_SLOT_ACTION_ID" \
  --mode sync \
  --input @slot-baseline.json
```

Compare the before and after provider payloads and their observation times. A changed result is correlated with the controlled change; it is not a connector explanation of cause. An unchanged result can reflect another availability layer, provider propagation, or provider-owned behavior.

Remove a conflict calendar safely

Removal is not an exploratory diagnostic step. Use it only when the intended state is to stop considering a known calendar and you have authoritative values for every selector.

Write `conflict-calendar-delete.json`. Note that `credentialId` is a string:

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

Submit the destructive operation with a new stable identity:

```
SH
export CALENDAR_DELETE_ACTION_ID='act_cal_availability_calendar_delete_001'
export CALENDAR_DELETE_KEY='cal-availability-calendar-delete-001'

aipctl \
  --native-bearer-token-file "$AIP_TOKEN_FILE" \
  action call "$AIP_URL" \
  cap:cal_diy:calendar.selected.delete \
  --action-id "$CALENDAR_DELETE_ACTION_ID" \
  --idempotency-key "$CALENDAR_DELETE_KEY" \
  --input @conflict-calendar-delete.json
```

The client sends every supplied supported field as a `DELETE` query parameter: `integration`, `externalId`, `credentialId`, and optional `delegationCredentialId`. Removal is high risk, approval-required, retry-unsafe, and has no rollback contract. Adding a calendar later is a new mutation, not restoration of the deleted provider state.

Diagnose a missing slot

Use this table after preserving the baseline:

Check	Evidence to compare	Safe next action
Event-type linkage	Actual `scheduleId` in provider output	Read that exact schedule
Weekly availability	Day and local start/end for the frozen range	Correct one reviewed interval if wrong
Dated override	Override date and local interval	Correct one override if wrong
Booking limits	Window, notice, duration, offset, and buffers	Review event-type configuration
Conflict selection	Exact integration, external id, and credential id	Add only the intended calendar
Busy time	Exact calendar set, range, time zone, and intervals	Inspect the overlapping provider event
Slot query	One selector branch, range, time zone, duration, and format	Correct the query and create a new baseline
Booking plan	Exact requested start and documented planner limits	Use it only for the final booking intent
Provider state	Credential health, response status, request id, and time	Escalate with retained evidence

The connector publishes no selected-calendar-specific read and no availability-explanation capability. If the remaining question requires either one, report that evidence gap instead of inventing a reason from nearby data.

Recover without replaying a mutation

State or failure	Safe response
`requires_human`	Complete trusted approval for the same action or let it expire
`in_flight`	Read durable status and wait; keep the same action and key
`outcome_unknown`	Reconcile the original provider operation; do not send a new key
Transport failure after possible dispatch	Treat the mutation as uncertain and reconcile
Authentication failure	Verify tenant, external-account binding, and credential revision
Provider conflict	Refresh the exact schedule or calendar evidence before reviewing a new intent
Rate limit or temporary provider failure on a read	Retry the read within deadline and policy
Invalid action	Correct the input and use a new action for the corrected intent
Idempotency collision	Stop; one key already owns different canonical input

Connector retries are supported for the reads in this guide, but not for its mutations. Never use a new action id or idempotency key merely to escape an uncertain mutation state.

Evidence checklist

Retain:

- catalog revision and capability schema and contract digests;
- tenant and external-account route, without copying provider credentials;
- frozen event-type, schedule, range, time zone, calendar, and slot coordinates;
- before and after event-type, schedule, calendar, busy-time, and slot results;
- action ids, one mutation idempotency key, approval evidence, and receipts;
- provider request and operation identifiers when present;
- observation times and any reported propagation or response-shape gap.

This evidence supports a bounded comparison. It does not prove live connector qualification, provider causality, future bookability, or successful booking.

Related documentation

- Slots and reservations
- Calendars and free/busy
- Schedules
- Event types and private links
- Error reference