

CHeKT Integration Guide for Third-Party Platforms and Devices

Raising CHeKT video alarms from your own detection logic, through
the CHeKT Public API

Version 0.20 · **Last updated** 2026-09-26

Contents

1. Purpose and Scope

1.1 Quickstart

2. Integration Options

Option 1 — Connect the device over ONVIF

Option 2 — Integrate with the CHeKT Cloud API (this guide)

Option 3 — Universal Cloud Receiver (future)

3. Architecture

4. Prerequisites

5. Authentication

How the dealer obtains an API key

Key handling requirements

6. What the Partner Must Expose to the Dealer

6.1 The dealer's CHeKT API key

6.2 Binding a rule to a CHeKT camera

7. Integration Flow

7.1 Discover the dealer's sites

7.2 Discover the cameras at a site

7.3 Store the mapping on the partner side

7.4 Trigger the alarm

7.5 Recording actions

7.6 Recovering the event and its video

7.7 Send a verification message

7.8 Identifiers — /ext, /ref and /txid

8. Behavior Partners Must Design For

8.1 Gates between an accepted event and a monitoring signal

8.2 Zone addressing can raise the alarm on a different camera

8.3 The alarm response carries no event ID

8.4 Media URLs expire after one hour

8.5 Clock accuracy

8.6 Recording is asynchronous

9. Error Handling and Rate Limits

9.1 Error envelope

9.2 Classify on code, discriminate on detail

9.3 Error codes and recommended handling

9.4 Rate limits

10. Reference

11. Integration Checklist

Next steps

Appendix A. Arming API

A.1 Two arming models

A.2 The endpoint families are mutually exclusive

A.3 Arming requests are all-or-nothing

A.4 Arming action request body

Appendix B. Question Index

1. Purpose and Scope

This guide describes how a third-party system integrates with CHeKT so that events generated by the partner's own logic are delivered as **video alarms** — reaching monitoring centers and end-user mobile apps through CHeKT's existing infrastructure.

Any platform that needs to produce video evidence, a mobile push notification, or an alarm signal for real-time monitoring can use this method, including:

- video analytics rule engines and behavioural AI systems
- on-premise AI appliances and edge devices
- access control systems, for critical incidents
- facial recognition systems

The integration does not depend on what kind of detection the partner performs. It only requires that the partner can call an HTTPS endpoint when its own rule becomes true. The partner keeps its own logic, rules and user interface; CHeKT provides alarm transport, video verification, monitoring-center distribution and the end-user experience.

1.1 Quickstart

The shortest path to a first video alarm. [§7](#) explains each step in full.

You need

- A dealer's CHeKT API key ([§5](#))
- A site with a Bridge installed and cameras registered ([§4](#))
- The site **armed** at the time of the test — a disarmed site rejects the event ([§8.1](#))

```
export CHEKT_API_KEY="your_key_here"
```

Step 1 — Find the site

```
curl -s "https://api.chekt.com/ext/v1/sites" \  
-H "Authorization: Bearer $CHEKT_API_KEY"
```

```
[ { "site_id": 40001, "account_reference_id": "ACME-0001",  
  "is_partition_enabled": 1, "is_activated": 1 } ]
```

Step 2 — Find the camera

```
curl -s "https://api.chekt.com/ext/v1/sites/40001/cameras" \  
-H "Authorization: Bearer $CHEKT_API_KEY"
```

```
[ { "device_id": "50001", "name": "Loading Dock", "status": "online",  
  "is_registered": 1, "zone_numbers": ["102"] } ]
```

`device_id` is the `{camera_id}` used in the next step.

Step 3 — Raise the alarm

Omitting the time fields is the simplest form — the server uses the time of receipt (§7.4).

```
curl -s -X POST \
  "https://api.chekt.com/ext/v1/sites/40001/cameras/50001/events" \
  -H "Authorization: Bearer $CHEKT_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{ "event_description": "Quickstart test [rule-1]" }'
```

```
{ "message": "OK" }
```

A single camera channel accepts one alarm request per 20 seconds (§7.4).

Step 4 — Confirm it

200 is an acknowledgement, not proof. Check the activity log over a time range around the call you just made:

```
NOW=$(date -u +%Y-%m-%dT%H:%M:%SZ)
AGO=$(date -u -v-10M +%Y-%m-%dT%H:%M:%SZ 2>/dev/null \
  || date -u -d '10 minutes ago' +%Y-%m-%dT%H:%M:%SZ)

curl -s -X POST "https://api.chekt.com/ext/v1/activity-logs/search" \
  -H "Authorization: Bearer $CHEKT_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{ "site_id": 40001, "category_id": 20007,
    "start_time": "\"$AGO\"", "end_time": "\"$NOW\"",
    "limit": 50, "sort_by": "event_time", "sort_order": "desc" }'
```

The row whose `detail.event_topic_raw` is `tns1:LocalAlarmReceiver/Quickstart test [rule-1]` is your event. Take its `event_id` and request the media:

```
curl -s -X POST "https://api.chekt.com/ext/v1/events-video-urls" \
  -H "Authorization: Bearer $CHEKT_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{ "event_ids": [900001] }'
```

If the response contains `mp4`, `snapshots` and `thumbnail` URLs, it worked. The URLs expire after one hour.

Next: what you must provide to the dealer (§6) · why an accepted event may not reach the monitoring center (§8.1) · error handling and rate limits (§9).

2. Integration Options

Three integration paths exist. Two are available today.

	Option 1 — ONVIF device	Option 2 — CHeKT Cloud API	Option 3 — Universal Cloud Receiver
Status	Available	Available — this guide	Future — not yet available
What the partner builds	ONVIF Profile S/T support on the device	An HTTPS API client, in the partner's cloud or on the on-site device	TBD
CHeKT Bridge required	Yes	Yes	No
How events reach CHeKT	ONVIF event topics, consumed by the Bridge	HTTPS call to CHeKT Cloud, which triggers the Bridge	Direct to CHeKT Cloud
Video source	The partner device, acting as a camera	Cameras already registered to the Bridge	TBD
Validation	Confirm the process with CHeKT	Partner self-test, then joint verification with CHeKT	TBD
Best for	Devices that are, or can behave as, a camera	Analytics platforms and AI appliances with an existing dealer base	TBD

Option 1 — Connect the device over ONVIF

If the partner's device supports ONVIF Profile S/T for video and delivers its analytics as ONVIF events, the CHeKT Bridge can register it in the same way as an IP camera, use its streams for video verification, and raise alarms from the event topics the Bridge supports. No CHeKT API development is required on the partner side.

Which profiles and event topics the Bridge supports should be confirmed with CHeKT.

Option 2 — Integrate with the CHeKT Cloud API (this guide)

The partner's analytics stay where they already run. When a rule fires, the partner calls CHeKT Cloud to trigger an event on a camera the CHeKT Bridge already manages, and CHeKT records and distributes the video alarm. No firmware work on the partner's device and no CHeKT-side development are required.

What calls the API is the partner's choice. It can be the partner's cloud, or the on-site AI device calling out over the internet. CHeKT sees the same HTTPS request either way, so a partner with no cloud of its own can still integrate. Where the on-site device calls directly, the key-handling requirements of [§5](#) apply to that device.

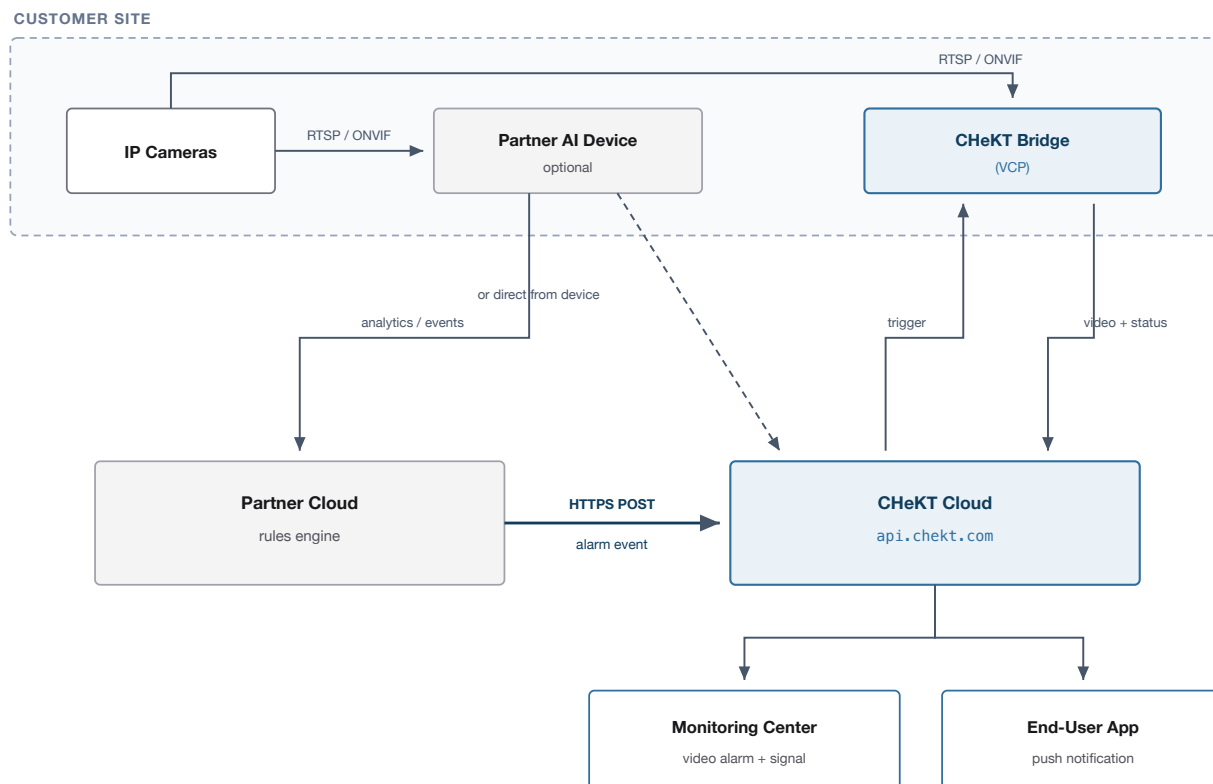
Use the cloud API, even for on-premise hardware. The Bridge's local API is not exposed to other devices on the site network, so a device on the same LAN cannot trigger it directly. The cloud endpoint is also a fixed address, whereas the Bridge's LAN address is normally DHCP-assigned. If you have a requirement that depends on local integration, raise it with CHeKT.

The cloud path needs internet connectivity at the moment of detection — which the monitoring service itself already requires. The trigger call returns no event ID; [§7.6](#) shows how to recover the event.

Option 3 — Universal Cloud Receiver (future)

CHeKT intends to offer a cloud receiver that accepts partner events for sites with no on-site Bridge. It is on the roadmap; no delivery date is being communicated. Build against Option 2 today — the authorization model, the site and camera mapping, and the event semantics in this guide are the same concepts the cloud receiver will use.

3. Architecture



- **The camera is shared.** The same camera streams to both the CHeKT Bridge and the partner's analytics over standard RTSP/ONVIF. CHeKT does not receive video from the partner.
- **The partner detects; CHeKT responds.** The partner's rules engine decides what is an alarm. CHeKT records the video, generates the alarm signal and distributes it.
- **The alarm references a CHeKT camera.** The API call names the CHeKT camera the alarm belongs to, and the video is taken from that camera.
- **CHeKT Cloud reaches the Bridge for you.** The partner never addresses the Bridge directly.

4. Prerequisites

All of the following must be true at the customer site before an alarm will succeed:

#	Requirement	Notes
1	The dealer has an active CHeKT dealer account	Integrations are authorized per dealer, not per partner
2	A CHeKT Bridge (VCP) is installed and online at the site	The Bridge produces the alarm video
3	The cameras to be alarmed are registered to that site	Only cameras in the site's camera list (§7.2) can be targeted
4	The site is activated in CHeKT	<code>is_activated: 1</code> in the site list (§7.1)
5	An API key is enabled on the dealer account	§5
6	The site is armed at the time of the event	§8.1 — the most common cause of rejected events

For partition-enabled sites, the Bridge must run firmware v6.0.0 or higher.

5. Authentication

Every request carries the dealer's API key as an HTTP Bearer token (JWT):

```
Authorization: Bearer <API_KEY>
```

How the dealer obtains an API key

The key is issued by the dealer from the dealer's own CHeKT account. The partner never requests keys from CHeKT directly.

1. Log in to the CHeKT Dealer Portal.
2. Open **Settings** (gear) → **Developer Settings** → **API Keys** — direct link: dealer.chekt.com/settings/api-keys. The account must hold the *Developer Tools* permission for this menu to appear.
3. Generate a key.

The **Last Used** column on the same screen shows whether calls made with that key are arriving — the quickest way to confirm authentication is working.

Key handling requirements

- **The key is shown once.** CHeKT cannot recover a lost key. Generate a new one and disable the old one.
- **The key is dealer-scoped.** It grants access to every site under that dealer, not a single site. Store it encrypted, server-side. Never embed it in a device image, a mobile app or client-side code.

6. What the Partner Must Expose to the Dealer

Every integration must meet two requirements. How they appear in the partner's product is the partner's design decision.

#	Requirement	Why
6.1	The dealer can supply — and later replace — their own CHeKT API key	Keys are issued per dealer account (§5); CHeKT does not issue keys to partners
6.2	Every rule that raises a CHeKT alarm resolves to exactly one CHeKT camera, chosen by the dealer	The alarm call addresses <code>{site_id, camera_id}</code> (§7.4), and nothing can derive that pairing automatically

A *rule* here is whatever configuration unit in the partner's product decides "these conditions should raise a CHeKT alarm" — rule, policy, trigger or scenario.

6.1 The dealer's CHeKT API key

The integration acts as the dealer. **Recommended:** validate the key on save with `GET /ext/v1/sites` (§7.1). A successful response confirms the key and returns the site list needed for 6.2.

6.2 Binding a rule to a CHeKT camera

Partner camera identifiers and CHeKT `camera_id` values are independent; neither side can match them automatically. The dealer makes the association once and the partner stores it (§7.3).

Recommended shape:

1. Show the dealer the list from `GET /ext/v1/sites/{site_id}/cameras` (§7.2), displaying each camera's `name`.
2. When the dealer picks one, store that camera's `device_id` together with the `site_id`.
3. When the rule fires, call the alarm endpoint with the stored `device_id` as `camera_id` (§7.4). You cannot call by name.

`name` is the label a person picks from; `device_id` is the key you call with. Letting a dealer type a numeric ID by hand produces mis-bindings that surface later as alarms on the wrong camera.

7. Integration Flow

7.1 Discover the dealer's sites

```
GET /ext/v1/sites?search={text}&limit={n}
Authorization: Bearer <API_KEY>
```

Both query parameters are optional. `search` matches the site's name and address text — not its ID; a numeric `site_id` passed to `search` returns no match. There is no lookup-by-ID on this endpoint, so search by name once, store the returned `site_id`, and use it for every later call. Every other endpoint takes the numeric `site_id`.

Response (abridged)

```
[
  {
    "dealer_id": 20001,
    "dealer_name": "Example Security",
    "site_id": 40001,
    "site_name": "Example Retail – Loading Dock",
    "address": "100 Example Street, Springfield",
    "account_reference_id": "ACME-0001",
    "account_number": "230601",
    "is_activated": 1,
    "monitoring_station_name": "Example Monitoring",
    "is_partition_enabled": 1,
    "partitions": [
      { "partition_id": 12, "partition_number": 1, "partition_name": "Front Office" },
      { "partition_id": 13, "partition_number": 2, "partition_name": "Warehouse" }
    ]
  }
]
```

Response fields

Field	Meaning	Use
<code>site_id</code>	CHeKT's numeric ID for the site	Used in the path of every later call. Store it
<code>site_name</code> , <code>address</code>	What the dealer recognizes	Show these when the dealer picks a site
<code>dealer_id</code> , <code>dealer_name</code>	The dealer who manages the site	Informational — the key already scopes you to the dealer
<code>is_activated</code>	Whether the site is active (<code>1</code> / <code>0</code>)	Do not offer <code>0</code> sites for rule binding
<code>account_number</code> , <code>account_reference_id</code>	The monitoring account number, and the dealer's own reference. Both optional — either can be <code>null</code>	Address the site through <code>/txid</code> and <code>/ref</code> (§7.8)
<code>monitoring_station_name</code>	The monitoring center receiving this site	Informational
<code>is_partition_enabled</code>	<code>1</code> = partition system, <code>0</code> = site system	Changes how arming behaves (Appendix A)
<code>partitions[]</code>	The site's partitions; empty on a site-system site	Only needed if you control arming (Appendix A)

At minimum, store `site_id` . There is no lookup by ID, so losing it means searching by name again.

7.2 Discover the cameras at a site

```
GET /ext/v1/sites/{site_id}/cameras
Authorization: Bearer <API_KEY>
```

```
[
  {
    "device_id": "50001",
    "mac_address": "AABBCCDDEEFF",
    "name": "Loading Dock",
    "device_type": "Camera",
    "status": "online",
    "status_updated_at": "2026-09-01T04:45:19.000Z",
    "is_registered": 1,
    "zone_numbers": ["1", "2"],
    "mjpeg_url": "https://api.chekt.com:443/api/v1/mjpeg?access_token={access_token}"
  }
]
```

- `device_id` is the `{camera_id}` used in the alarm call. It is returned as a string here, but as a number in the activity log (§7.6) — normalise before comparing.
- `name` is what the dealer recognizes — use it as the picker label.
- `status` and `is_registered` let you hide offline or unregistered cameras.

A zone-oriented view, `GET /ext/v1/sites/{site_id}/zones` , returns `zone_numbers[]` per device.

About zone numbers. CHeKT assigns a zone number automatically when a camera is registered at a site. Bridges carry one too, but only camera zones can raise an event. The zone number is what identifies the point in the signal sent to the monitoring center, whether the alarm was raised by zone or by camera (§7.4).

One camera can hold several zone numbers, and several cameras can hold the same zone number, so a zone number does not by itself resolve to one camera. Address alarms by `camera_id` . If you must build on zones, read §8.2 first.

7.3 Store the mapping on the partner side

CHeKT does not store the partner-camera ↔ CHeKT-camera mapping and exposes no endpoint to register it. What the dealer binds in §6.2 has to live in the partner's own storage.

Store it in your own database, next to each rule. The alarm call (§7.4) needs exactly two values, `site_id` and `camera_id` , so each rule must be able to supply both:

Your rule	site_id	camera_id
rule-8821 — Person detected, Loading Dock	40001	50001
rule-8822 — Loitering, Front Entrance	40001	50002

The table layout and column names are up to you; CHeKT only sees the two values when you call it.

Refresh the site and camera lists periodically and re-validate the bindings. Dealers add, rename and delete cameras, and a binding that points at a deleted camera fails at alarm time.

7.4 Trigger the alarm

This call creates the video alarm and, where the site is configured for it, sends the signal to the monitoring center.

```
POST /ext/v1/sites/{site_id}/cameras/{camera_id}/events
Authorization: Bearer <API_KEY>
Content-Type: application/json
```

Request body — all fields optional

```
{
  "event_description": "Person detected [rule-8821]",
  "event_time": "2026-09-23T14:10:28.923+00:00",
  "accessory_current_time": "2026-09-23T14:10:30.111+00:00"
}
```

Field	Type	Required	Description
<code>event_description</code>	string	no	Free text shown to the monitoring operator. Carried verbatim into the event topic as <code>tns1:LocalAlarmReceiver/<event_description></code> , so it doubles as your correlation token (§7.6). A descriptive value is the difference between "Alarm" and "Person detected [rule-8821]" on the operator's screen.
<code>event_time</code>	string (RFC3339)	no	When the partner's rule fired. Recorded as the event's timestamp and checked for staleness. If omitted, the server uses the time of receipt.

Field	Type	Required	Description
accessory_current_time	string (RFC3339)	yes, when event_time is sent	The partner device's own clock at send time, used for clock-skew diagnosis.

Length and encoding

Keep event_description to 40 bytes or less. The activity log stores it as tns1:LocalAlarmReceiver/<event_description> in a 64-byte field, and the 24-byte prefix leaves 40 bytes for your text. Anything longer is silently truncated — the call still returns 200 , and nothing in the response says the text was cut.

The cut is by bytes, not characters, and it is not UTF-8 aware:

Sent	Stored in event_topic_raw
Person detected [rule-8821] — 27 bytes	kept whole
1 KB of ASCII	the first 40 characters
44 Korean characters — 132 bytes	the first 13 characters; the 14th comes back as a broken byte
"" , or the field omitted	active

Because the text is cut from the end, a correlation token (§7.6) survives only if the whole value fits — keep the value short rather than moving the token to the front. / , quotes, backslashes and < > & | are all accepted and round-trip unchanged.

Response

```
{ "message": "OK" }
```

Time fields

Recommended: omit both time fields. The server then uses the time of receipt, and neither the format rule nor the staleness rule below can affect you. Send them only when the detection time differs meaningfully from the send time.

Format. Both fields must be RFC3339 with three-digit milliseconds and a numeric offset:

Value	Result
2026-09-23T14:10:28.923+00:00	Accepted
2026-09-23T14:10:28.923Z	Rejected — the Z suffix is not accepted
2026-09-23T14:10:28.923456+00:00	Rejected — six-digit microseconds
event_time without accessory_current_time	Rejected

```
400 { "error": { "code": 19001, "message": "Camera event error",
  "detail": "ERR_PARSE: invalid event_time: it supports only RFC3339 date-times(e.g. 2006-01-02T15:04:05.000-07:00)" } }

400 { "error": { "code": 19001, "message": "Camera event error",
  "detail": "ERR_PARSE: accessory_current_time always needs to be provided when event_time is provided" } }
```

This restriction applies to the event endpoints only. `start_time` / `end_time` on the activity-log search (§7.6) accept the `Z` suffix and omitted milliseconds. Take care if you share one timestamp helper between them.

Staleness. An `event_time` outside the accepted window returns `200` but produces no event and no activity-log rows — the response alone will not tell you. The window is a matter of minutes and is not a published figure, so do not compute against it: send the time of detection, and if your design queues events, drop stale entries instead of sending them. To retrieve footage from a period that has already passed, use the playback action (§7.5).

Per-channel call rate

One camera channel accepts one request per 20 seconds. Exceeding it is rejected with `400` / `19001` — not `429` :

```
400 { "error": { "code": 19001, "message": "Camera event error",
  "detail": "Too Many Requests for 00: '1 request / 20 second / channel' is allowed" } }
```

A client that treats only `429` as a rate limit will misread this as a camera failure. Discriminate on `detail` (§9.2).

Example (with time fields)

```
curl -X POST "https://api.chekt.com/ext/v1/sites/40001/cameras/50001/events" \
-H "Authorization: Bearer $CHEKT_API_KEY" \
-H "Content-Type: application/json" \
-d "{
  \"event_description\": \"Person detected [rule-8821]\",
  \"event_time\": \"$(date -u +%Y-%m-%dT%H:%M:%S).000+00:00\",
  \"accessory_current_time\": \"$(date -u +%Y-%m-%dT%H:%M:%S).000+00:00\"
}"
```

Zone-addressed alternative

```
POST /ext/v1/sites/{site_id}/zones/{zone_number}/events
```

Same body and response. This endpoint is also documented in CHekT's support article *Creating Events and Alarms Using Cloud APIs* (§10), and it works. This guide recommends `camera_id` instead: several cameras can hold the same zone number, and an event sent to it is raised on the camera with the lowest `camera_id`. The call still returns `200`, so the response alone will not tell you it went to a different camera (§8.2). Where every zone number on a site maps to exactly one camera, the two are equivalent.

What an accepted alarm produces

An accepted alarm produces several activity-log rows that share one `event_id` :

Category	trigger	What it is
20005 Sensor	LE API	The origin record — the only row that marks the event as API-originated
20023 Signal	Sensor	The signal transmitted to the monitoring center. Conditional — see below
20007 Video event	Alarm / Bridge	Recording lifecycle: snapshot recording → snapshots uploaded → video recording → video uploaded

A monitoring signal is sent only where the dealer has enabled it. The setting is in the Dealer Portal under **Site** → **Settings** → **Alarm Library** → **Bridge Alarm Input Codes**, as an ENABLE checkbox per zone type (Entry/Exit, Follower, Instant, 24 Hour, Exterior and so on). Where the target camera's zone type is not enabled, the alarm is accepted and video is produced, but no 20023 row appears. Before verifying a first alarm, ask the dealer which zone type the target camera uses and whether it is enabled.

The Live Video Only zone type ignores API alarms. It is for cameras that should not create events at all. An alarm sent to such a camera returns 200 but produces no event, no video and no signal.

Where a signal is sent, the Signal row reports what was transmitted:

```
{
  "category_id": 20023, "status": "alarm",
  "zone_number": "141", "account_number": "230601",
  "event_id": 900002,
  "detail": { "event_code": "E131", "send_status": "SUCCESS" }
}
```

- event_code is a Contact ID code — E131 for the alarm, R131 for its restore.
- send_status reports the outcome — use it to answer "did the alarm go out?" without a support ticket.
- zone_number and account_number are how the monitoring center identifies the point and the account, even when the alarm was raised by camera_id.

Repeated alarms on one zone can be bypassed. Where a zone alarms repeatedly, CHeKT marks it **swinger bypassed** and further alarms from that zone stop. The trigger call still returns 200. The bypass appears only in the activity log, as its own event:

```
20005 Camera swinger bypassed eid=900003 zone=141
20023 Sensor swinger bypassed eid=900003
{"event_code":"E575","send_status":"SUCCESS"}
```

The threshold and current count are not exposed through the API. Send one alarm per incident, not one per detection (§8.6).

7.5 Recording actions

Two actions produce video without raising an alarm. Both take no request body, return { "message": "OK" }, and send nothing to the monitoring center.

Action	What it produces
POST /ext/v1/sites/{site_id}/cameras/{camera_id}/actions/create-recording-event	A live recording starting at the moment of the call — the same recording as §7.4, without the alarm

Action	What it produces
<code>POST /ext/v1/sites/{site_id}/cameras/{camera_id}/actions/create-recording-playback</code>	A two-minute clip from stored video — the only way to retrieve footage from a period that has already passed

Zone-addressed equivalents exist at `POST /ext/v1/sites/{site_id}/zones/{zone_number}/actions/create-recording-event` and `.../create-recording-playback`, with the same ambiguity as zone-addressed alarms (§8.2).

7.6 Recovering the event and its video

The alarm response carries no event ID (§8.3), so retrieving the video means calling two endpoints in order:

Step	Endpoint	What it does
a	<code>POST /ext/v1/activity-logs/search</code>	Finds the event you raised and gives you its <code>event_id</code>
b	<code>POST /ext/v1/events-video-urls</code>	Returns the mp4 and snapshot URLs for that <code>event_id</code>

a. Find your event

One query on `category_id: 20007` (Video event) both identifies the event and tells you when the media is ready.

```
POST /ext/v1/activity-logs/search

{
  "start_time": "<window start>",
  "end_time": "<window end>",
  "category_id": 20007,
  "limit": 200,
  "sort_by": "event_time",
  "sort_order": "desc"
}
```

Identify your events by `detail.event_topic_raw`. The `event_description` you sent in §7.4 comes back after the prefix, verbatim:

```
what you sent (§7.4)
{ "event_description": "Person detected [rule-8821]" }

what the activity log returns
"detail": { "event_topic_raw": "tns1:LocalAlarmReceiver/Person detected [rule-8821]" }
```

Rows starting with `tns1:LocalAlarmReceiver/` are yours, and the string after it tells you which rule fired. The Bridge's own detections start with `tns1:VideoSource/...` or `RuleEngine/...`, so the two never mix.

```
eid=900001 snapshot recording    tns1:LocalAlarmReceiver/Person detected [rule-8821]    ← yours
eid=900001 video uploaded      (no detail)                                           ← same event, media
ready
eid=900006 video recording      tns1:VideoSource/MotionAlarm                               ← the Bridge's own
detection
```

1. Filter on the `tns1:LocalAlarmReceiver/` prefix and take the `event_id`.
2. When a row with the same `event_id` reaches `status: "video uploaded"`, the media is ready. That row carries no `detail`, so match it on `event_id`.
3. Request media URLs for that `event_id` (step b).

Use `event_description` as a correlation token. Whatever you put in it comes back in `event_topic_raw`, so you can find your own events without guessing from time windows. A rule identifier alone tells you *which rule* fired; to pin down *one call*, add a value that is unique per call — for example `"Person detected [r8821/7f3a]"`. The whole value has to stay within 40 bytes (§7.4) or the token is cut off. It is also shown to the operator, so keep it human-readable.

Filter on `20007`, not `22003` (Alarm). The names suggest otherwise, but only `20007` always carries an `event_id`:

category_id	Name	Carries event_id ?
20007	Video event	Always — use this one
22003	Alarm	Never
20005	Sensor	Sometimes
20023	Signal	Sometimes
20001	Arming	Sometimes

The full list is available from `GET /ext/v1/activity-logs/categories`. A `20007` row looks like this:

```
{
  "id": "01a0aea6d2e270fabe1b0f4de2671e8a",
  "site_id": 40002,
  "zone_number": "131",
  "event_id": 900005,
  "event_time": "2026-09-17T09:16:03.671Z",
  "category_id": 20007,
  "trigger": "Bridge",
  "status": "video uploaded",
  "device_id": 50002,
  "device_name": "Lobby PTZ",
  "detail": { "event_topic_raw": "RuleEngine/LineDetector/Crossed" }
}
```

Query parameters. Only `start_time` is required. Sent on its own it returns rows for **every site under the dealer and every category**, up to the default `end_time` of now, newest first, with a default `limit` of 1000 — bridge motion, audio playback and arming rows from other sites included. To get only your own events, narrow to `category_id: 20007` and then filter on the `tns1:LocalAlarmReceiver/` prefix in `detail.event_topic_raw`. `limit` accepts up to 1000. `site_id`, `zone_number` and `name` (a prefix match on site and device names) are optional filters. A `site_ids` array is not accepted.

Sweep dealer-wide. Omitting `site_id` returns rows for every site under the dealer, each carrying its own `site_id` and `site_name`. This endpoint is rate-limited per API key, not per site (§9.4), so one dealer-wide query is far cheaper than one query per site.

Paginate with the cursor. The response carries a `page` object. Copy its `cursor` into the **top level** of the next request body and repeat while `has_more` is `true`. Sending it back nested inside a `page` object is rejected (`20101`, "page" is not

allowed), as are `offset` and page numbers — the log is append-only, so an offset would skip or repeat rows as new events arrive.

```
"page": { "limit": 200, "cursor": "WzE30TAXMzI1NzIyMjUsLi4u", "has_more": true }
```

b. Request the media URLs

```
POST /ext/v1/events-video-urls

{ "event_ids": [900001, 900005] }
```

Returns `mp4`, `thumbnail` and `snapshots[]` as presigned URLs, plus `expires_in: 3600`. The URLs expire after one hour — fetch and store the media, or re-request on demand. Do not persist the URLs themselves.

7.7 Send a verification message

Prerequisite — the site must have at least one contact. A contact created through the public Contact API (`POST /ext/v1/sites/{site_id}/contacts`) is eligible; no separate Dealer Portal registration is required. Calling on a site with no contacts returns `21001`.

Two endpoints, differing only in whether a specific event is named:

```
POST /ext/v1/events/{event_id}/verifications      # verification for one event
POST /ext/v1/verifications                        # site-level verification
```

Field	Type	Required	Description
<code>user</code>	string	yes	Identifies who raised the request. Recorded on the <code>20008</code> activity-log row; free text, not validated
<code>monitoring_portal_url</code>	string	no	A link to include for the recipient. Used with the site-level call

```
{
  "user": "operator-01",
  "monitoring_portal_url": "https://monitoring.chekt.com?site=40001&zone=102&event=900001"
}
```

Returns `{ "verification_id": "25278" }`. Delivery appears in the activity log as a `20008` Verification row with `status: sent`.

If you create contacts through the API, `external_contact_id` is your own identifier for each. It is unique per dealer, not per site — reusing a value that exists anywhere under that dealer fails the create. Contacts entered in the Dealer Portal have it as `null` and cannot be addressed through `/ref`.

7.8 Identifiers — /ext, /ref and /txid

The same endpoints exist under three path prefixes, differing only in which identifier names the site:

Prefix	Site named by	Who sets it
/ext/v1/sites/{site_id}	site_id — CHeKT's internal ID	CHeKT. Always present
/ref/v1/sites/{account_reference_id}	The dealer's own reference for the account	The dealer. Optional
/txid/v1/sites/{account_number}	The monitoring account number	The dealer. Optional. Arming endpoints only

Use /ext . It is the only prefix that covers the whole endpoint surface, and `site_id` is the only identifier every site is guaranteed to have. The other two are optional, not interchangeable (a site can carry different values in each), and `null` on some sites — for those sites `/ref` and `/txid` cannot address them at all. Under `/ref` , triggering is available by zone only — with the ambiguity described in §8.2:

Endpoint	/ext	/ref
GET .../cameras	Yes	Yes
POST .../cameras/{camera_id}/events and .../actions/*	Yes	No — 404 / 10400
POST .../zones/{zone_number}/events and .../actions/*	Yes	Yes

The wrong prefix looks like a missing site. A CHeKT `site_id` passed to a `/ref` path is read as an `account_reference_id` :

```
GET /ref/v1/sites/40001/arming
404 { "error": { "code": 10403, "message": "Site not found",
    "detail": "dealer ID=<id>, account_reference_id=40001, result length=0" } }

GET /ext/v1/sites/40001/arming      → 200
```

If a site you know exists comes back as not found, check the prefix before the ID. `detail` names the identifier the lookup actually used.

8. Behavior Partners Must Design For

8.1 Gates between an accepted event and a monitoring signal

A `200` on the trigger call means the event was *accepted*. It does not mean an alarm reached the monitoring center. Several independent gates sit in between, and most of them act silently, after the `200` has been returned.

Gate	Applies when	API response	Video	Signal	How you can tell
1 Arming	Site or partition disarmed, <code>DisarmedActivityEnabled</code> off	<code>400 19001 / 20000</code>	none	none	The error response
1' Disarmed activity	Disarmed, <code>DisarmedActivityEnabled</code> on	<code>200</code>	produced	none	A <code>20020</code> row
2 AI filtering	CHeKT-side AI filtering enabled on the camera and the clip does not match	<code>200</code>	produced	none	No <code>20023</code> row

Gate	Applies when	API response	Video	Signal	How you can tell
3 Signal routing	The camera's zone type is not enabled in the Alarm Library (§7.4)	200	produced	none	No 20023 row
3' Live Video Only	The camera's zone type is Live Video Only (§7.4)	200	none	none	No rows at all
4 Swinger bypass	The zone was bypassed after repeated alarms (§7.4)	200	—	none	A swinger bypassed row
5 Staleness	event_time sent is outside the accepted window (§7.4)	200	none	none	No rows at all
6 Channel throttle	A repeat call on the same camera channel within 20 s	400 19001	none	none	The error detail

One principle covers all of them: 200 is an acknowledgement. The only evidence that a monitoring signal was sent is the 20023 row in the activity log (§7.4).

Gate 1 — arming state

The partner's analytics run continuously; CHeKT raises alarms only while the site — or, on a partition site, the camera's partition — is armed. A daytime test against a disarmed site will reject every event. The partner does not need to mirror CHeKT arming state: expect rejections outside armed hours and handle them quietly (§9.3).

The rejection is HTTP 400 with the upstream 406 embedded in detail. The zone endpoint returns the same shape under code 20000 ("Zone event error").

```
{
  "error": {
    "code": 19001,
    "message": "Camera event error",
    "detail": "406 - {\"message\":\"Not in condition to make an event - IsArmed: false, ZoneType: instant, DisarmedActivityEnabled: false\"}"
  }
}
```

- IsArmed — the arming state of the site, or of the partition the camera belongs to
- ZoneType — the zone's configured type
- DisarmedActivityEnabled — whether the zone records activity while disarmed. A dealer setting, not an API parameter

With DisarmedActivityEnabled on, an event on a disarmed site is no longer rejected, but it does not become an alarm: it is recorded as a 20020 Disarmed Event with video, and no 20023 Signal is produced. If you need video outside armed hours, ask the dealer to enable it.

```
20020 LE API activity eid=900004
20020 Activity video recording
20007 Bridge video uploaded
(no 20023 Signal)
```

Arming state changes on its own. Sites arm and disarm on schedules, so a site read as disarmed may be armed minutes later. If you read arming state for troubleshooting, read it immediately before acting: GET /ext/v1/sites/{site_id}/arming

on site-system sites, `GET /ext/v1/sites/{site_id}/partition-arming` on partition sites. Calling the wrong one is rejected by design — see Appendix A for the two arming models.

Do not arm or disarm a site as a side effect of an analytic rule. Arming authority belongs to the dealer and the end user. If your product does control arming, read Appendix A first.

Gate 2 — AI filtering

Where CHekT-side AI filtering is enabled on a camera, CHekT analyses the recorded clip before the event becomes an alarm. If nothing matches, the event is marked as an AI failure and no signal is sent. The call still returned `200`, and the video event and clip still exist — only the `20023` row is absent.

This matters for integrations of this kind, because the partner's product is already the detector: two detectors in series must both agree before the monitoring center is notified.

- The camera object ([§7.2](#)) exposes no AI configuration field. Confirm the setting with the dealer.
- Where the partner supplies the detection, CHekT-side filtering on those cameras is usually redundant. Agree the setting with the dealer before joint testing.

8.2 Zone addressing can raise the alarm on a different camera

One camera can hold several zone numbers, and several cameras can hold the same zone number. In the second case, an event sent to that zone is raised on the camera with the **lowest** `camera_id`.

The call does not fail. It returns `200` and an event is created, so the only way to discover that it went to a different camera is to read the activity log. On a site where two cameras share one zone number:

```
POST .../zones/171/events      → 200  event raised on camera A    (A's own zone)
POST .../zones/919/events      → 200  event raised on camera B    ← the zone A and B share
POST .../cameras/{A}/events    → 200  event raised on camera A
```

The second line is the problem: the partner believes it alarmed camera A, while the operator is looking at camera B's video.

Addressing by `camera_id` avoids this entirely — a `camera_id` is unique per camera, so the target is never ambiguous.

If you must build on zones, check the `GET /ext/v1/sites/{site_id}/zones` response for the same zone number under more than one `device_id`. Where it appears, that zone resolves to the lowest `camera_id`. The published specification describes a `zone_as_id` field for this purpose, but it is not present in the current `/zones` response, so the response does not state the resolution directly.

8.3 The alarm response carries no event ID

`POST .../events` returns `{ "message": "OK" }` and no identifier of any kind. To recover the event, search the activity log ([§7.6](#)): filter on `category_id: 20007` and identify your events by the `tns1:LocalAlarmReceiver/` prefix in `detail.event_topic_raw`. Put a rule identifier in `event_description` for one-to-one matching — within the 40-byte limit ([§7.4](#)), or it is truncated away — and reconcile asynchronously rather than inline.

8.4 Media URLs expire after one hour

Presigned media URLs ([§7.6](#)) carry `expires_in: 3600`. Re-request them on demand rather than caching them.

8.5 Clock accuracy

If you send `event_time` and `accessory_current_time`, they are only as useful as the partner device's clock. Keep the device synchronized to NTP. Clock drift between the partner device, the Bridge and the camera is a recurring source of "the

video does not match the event" reports. Omitting both fields (§7.4) avoids the problem on the partner side.

8.6 Recording is asynchronous

An alarm or recording call returns as soon as CHeKT accepts it. The Bridge then records, packages and uploads the media, which takes appreciably longer. A `200` means "accepted", not "the clip exists". If your workflow depends on the media, confirm it through the activity log (§7.6) before acting on it.

Concurrency. Simultaneous requests to different cameras on one Bridge are all accepted and all deliver media — six concurrent requests across three cameras produce six clips. There is no per-Bridge ceiling to design around at that scale. The per-channel limit of one request per 20 seconds still applies (§7.4).

Send one alarm per incident, with a descriptive `event_description` — not one per frame or per detection. A rule that fires repeatedly on one scene produces clips no operator will watch, and can trigger a swinger bypass (§7.4).

9. Error Handling and Rate Limits

9.1 Error envelope

Every error is wrapped in an `error` object. `code` is a CHeKT-specific numeric code, independent of the HTTP status.

```
{
  "error": {
    "code": 19001,
    "message": "Camera event error",
    "detail": "406 - {\"message\": \"Not in condition to make an event - IsArmed: false, ZoneType: instant, DisarmedActivityEnabled: false\"}"
  }
}
```

Arming requests add an `errors` array inside `error` (Appendix A):

```
{
  "error": {
    "code": 13001,
    "message": "Partitions not found for the given site.",
    "detail": "10423 Partitions not found for the given site. partition_ids=999999",
    "errors": [
      { "error_code": 10423, "message": "Partitions not found for the given site.",
        "partition_ids": [999999] }
    ]
  }
}
```

9.2 Classify on code, discriminate on detail

`code` alone is not always enough. `19001` (camera event) and `20000` (zone event) each cover at least three situations that need different handling:

detail	Meaning	What to do
contains <code>Not in condition to make an event</code>	Site or partition disarmed (§8.1)	Do not retry. Log quietly
contains <code>ERR_PARSE:</code>	Malformed request (§7.4)	Do not retry. Fix the request
contains <code>Too Many Requests for</code>	Per-channel rate exceeded (§7.4)	Retry after 20 seconds

`detail` sometimes wraps an upstream response verbatim, so its prefix is not consistent — the disarmed rejection starts with `406 - {...}`. **Match on substring containment, not on a prefix.** Log `detail` verbatim for support.

9.3 Error codes and recommended handling

Code	HTTP	Meaning
Authentication		
<code>10002</code>	401	No <code>Authorization</code> header
<code>10003</code>	401	Wrong authorization scheme — it must be <code>Bearer</code>
<code>10004</code>	401	API key expired
<code>10005</code>	401	API key disabled in the Dealer Portal
Target not found		
<code>10400</code>	404	URL does not exist — a mistyped path, or an endpoint the prefix does not carry (§7.8)
<code>10401</code>	404	No site for this dealer
<code>10403</code>	404	No site with this <code>account_reference_id</code> — seen on <code>/ref</code> (§7.8)
<code>10404</code>	404	No site with this <code>site_id</code>
<code>10411</code>	404	No camera for this <code>site_id</code> / <code>camera_id</code>
<code>10417</code>	404	No device for this <code>site_id</code> / <code>zone_number</code>
<code>10423</code>	404	The requested partitions do not belong to this site
Request rejected		
<code>19001</code>	400	Camera event failed — reason in <code>detail</code> (§9.2)
<code>20000</code>	400	Zone event failed — reason in <code>detail</code> (§9.2)
<code>13001</code>	400 / 403	Arming request rejected — reasons in <code>error.errors[]</code> (Appendix A)
<code>10301</code>	403	Site-level arming endpoint called on a partition-system site
<code>10303</code>	403	Partition arming endpoint called on a site-system site
<code>16001</code>	400	Camera lookup failed
<code>20001</code>	400	Site lookup failed
<code>20101</code>	400	Activity-log search parameter validation failed (§7.6)
<code>20201</code>	400	Media URL request parameter validation failed
<code>20202</code>	403	The request included an <code>event_id</code> you cannot access

Code	HTTP	Meaning
21001	400	Verification message failed — including when the site has no contacts (§7.7)
24001	400	Two-minute playback creation failed
10151	429	Rate limit exceeded (§9.4)

This list is not exhaustive. Handle unknown codes generically rather than switching exhaustively on the ones above.

Recommended client behavior

Condition	Action
401	Do not retry. Tell the dealer "CHeKT API key invalid" and prompt for re-entry
403 with 10301 or 10303	Not a key problem. Wrong arming endpoint family for this site type — re-read <code>is_partition_enabled</code> and call the other family
13001 (400 or 403)	Not a key problem. The arming request was rejected; reasons are in <code>error.errors[]</code>
Any other 403	Do not retry. Ask the dealer to check the key's permissions
Site or camera not found	Do not retry. The binding is stale — refresh §7.1 / §7.2 and flag the rule
Event rejected because disarmed	Do not retry. Expected behavior, not a fault. Log it; do not alert the dealer
Channel throttle	Retry once after 20 seconds
5xx / network timeout	Retry with capped exponential backoff. Alarms are time-critical — one that arrives ten minutes late is worse than one that failed fast and was logged
Sustained failure	Surface integration health in the partner UI, so the dealer sees the integration is down before an incident does

The alarm call is not idempotent. A retry after an ambiguous timeout can produce a duplicate alarm at the monitoring center. Prefer a single retry within a short window, then log.

9.4 Rate limits

Limits are enforced **per API key and per endpoint group** — not per site. Every response carries the current state:

```
X-RateLimit-Limit: 6
X-RateLimit-Remaining: 0
X-RateLimit-Reset: 1790131572
Retry-After: 60
```

Endpoint	Current limit
POST <code>/ext/v1/activity-logs/search</code>	6 per 60 s — its own bucket
GET <code>/ext/v1/sites</code> , <code>/cameras</code> , <code>/zones</code> , <code>/activity-logs/categories</code> , POST <code>/ext/v1/events-video-urls</code>	200 per window of roughly 10 s — shared bucket
POST <code>.../cameras/{camera_id}/events</code>	1 per 20 s per camera channel — returns 400 19001 , not 429 (§7.4)

Exceeding a bucket limit returns `429` with a normal error envelope:

```
{ "error": { "code": 10151, "message": "Too many requests", "detail": "Ratelimit id(2)" } }
```

Read `X-RateLimit-Remaining` and honour `Retry-After`; do not retry a `429` immediately. The activity-log limit is the one that shapes design, because it is how you recover events (§7.6): changing `site_id` does not give you a fresh allowance, so query dealer-wide and paginate with the cursor.

These figures describe the current mechanism and are not a formal contract. If you expect volume beyond ordinary alarm traffic — bulk enumeration across a large dealer base, for example — raise it with CHeKT before going live.

10. Reference

The Swagger reference is the authoritative definition of request and response shapes — follow it for field names, types and paths. This guide explains how the pieces fit together and how the platform behaves; it does not replace the spec.

One known difference: the `zone_as_id` field described in the spec is not present in the `/zones` response (§8.2). Design to this guide on that point; everywhere else, the spec governs.

Resource	Location
CHeKT Public API reference (Swagger)	https://public-apidoc-chekt.web.app/
Creating Events and Alarms Using Cloud APIs (support article)	support.chekt.com/.../creating-events-and-alarms-using-cloud-apis
API key management	https://dealer.chekt.com/settings/api-keys
API base URL	https://api.chekt.com

Endpoint summary — every `/ext` endpoint in the published spec

Purpose	Method	Path	In this guide
Sites and devices			
List sites	GET	<code>/ext/v1/sites</code>	§7.1
List cameras at a site	GET	<code>/ext/v1/sites/{site_id}/cameras</code>	§7.2
List zones at a site	GET	<code>/ext/v1/sites/{site_id}/zones</code>	§7.2
List bridges at a site	GET	<code>/ext/v1/sites/{site_id}/bridges</code>	—
List MP directions	GET	<code>/ext/v1/sites/{site_id}/mp-directions</code>	Spec only
Alarms and recording			
Trigger alarm (camera)	POST	<code>/ext/v1/sites/{site_id}/cameras/{camera_id}/events</code>	§7.4
Trigger alarm (zone)	POST	<code>/ext/v1/sites/{site_id}/zones/{zone_number}/events</code>	§7.4 , §8.2
Record without alarm (camera)	POST	<code>/ext/v1/sites/{site_id}/cameras/{camera_id}/actions/create-recording-event</code>	§7.5

Purpose	Method	Path	In this guide
Two-minute playback clip (camera)	POST	<code>/ext/v1/sites/{site_id}/cameras/{camera_id}/actions/create-recording-playback</code>	§7.5
Record without alarm (zone)	POST	<code>/ext/v1/sites/{site_id}/zones/{zone_number}/actions/create-recording-event</code>	§7.5
Two-minute playback clip (zone)	POST	<code>/ext/v1/sites/{site_id}/zones/{zone_number}/actions/create-recording-playback</code>	§7.5
Events and media			
Search activity logs	POST	<code>/ext/v1/activity-logs/search</code>	§7.6
Activity-log categories	GET	<code>/ext/v1/activity-logs/categories</code>	§7.6
Get event media URLs	POST	<code>/ext/v1/events-video-urls</code>	§7.6
Verification and contacts			
Send verification (by event)	POST	<code>/ext/v1/events/{event_id}/verifications</code>	§7.7
Send verification (site-level)	POST	<code>/ext/v1/verifications</code>	§7.7
Create / list contacts	POST / GET	<code>/ext/v1/sites/{site_id}/contacts</code>	§7.7
Get / update / delete a contact	GET / PUT / DELETE	<code>/ext/v1/sites/{site_id}/contacts/{contact_id}</code>	Spec only
Contact validation	POST	<code>/ext/v1/sites/{site_id}/contacts/{contact_id}/validation</code>	Spec only
Arming			
Read arming state	GET	<code>/ext/v1/sites/{site_id}/arming</code>	Appendix A
Arm / disarm site	POST	<code>/ext/v1/sites/{site_id}/arming/actions/arm , .../disarm</code>	Appendix A
Read partition arming state	GET	<code>/ext/v1/sites/{site_id}/partition-arming</code>	Appendix A
Arm / disarm partitions	POST	<code>/ext/v1/sites/{site_id}/partition-arming/actions/arm , .../disarm</code>	Appendix A
Audio			
List audio messages	GET	<code>/ext/v1/sites/{site_id}/audio-messages</code>	Spec only
Play an audio message	POST	<code>/ext/v1/sites/{site_id}/audio-messages/{audio_message_id}/play</code>	Spec only
List audio devices	GET	<code>/ext/v1/sites/{site_id}/audio-devices</code>	Spec only
Play audio	POST	<code>/ext/v1/sites/{site_id}/audio-play</code>	Spec only

"Spec only" endpoints are outside the alarm integration this guide covers; see the Swagger reference for their request and response shapes. The `/ref` prefix carries the same site, contact, arming, audio, camera-list and zone endpoints; `/txid` carries the arming endpoints only (§7.8).

11. Integration Checklist

Self-verify against this list before requesting joint testing with CHeKT.

Dealer setup

- ☐ The dealer can supply and later replace their CHeKT API key
- ☐ The key is validated on save via `GET /ext/v1/sites`
- ☐ The key is stored encrypted, server-side, and never exposed client-side
- ☐ Deactivated sites are excluded from the site list
- ☐ CHeKT cameras are offered by `name`, not entered as raw IDs
- ☐ Each alarm-producing rule is bound to exactly one `{site_id, camera_id}`
- ☐ Bindings are re-validated on a schedule and on failure

Alarm call

- ☐ A meaningful `event_description` is sent, with a rule identifier for correlation
- ☐ Time fields are either omitted, or `event_time` and `accessory_current_time` are both sent from an NTP-synchronized clock in the required format (§7.4)
- ☐ Calls respect the one-per-20-seconds-per-camera-channel limit
- ☐ One alarm per incident, not one per frame or detection (§8.6)
- ☐ No duplicate alarms under retry

Behavior

- ☐ Armed site: the event produces a video alarm at the monitoring portal, confirmed by a `20023` row
- ☐ Disarmed site: the rejection is handled quietly and logged, not retried
- ☐ `19001` / `20000` are discriminated on `detail` (§9.2)
- ☐ Offline camera: the failure is surfaced to the dealer
- ☐ Invalid key: the dealer is prompted to re-enter it
- ☐ Where the workflow depends on the clip, its presence is confirmed rather than assumed (§7.6, §8.6)
- ☐ If arming is controlled: partition-enabled sites use the Partition Arming endpoints (Appendix A)
- ☐ If verification messages are used: the target site has at least one contact (§7.7)

Next steps

1. **Request the certification round.** Contact CHeKT with the integration option you built against, how you meet the two requirements of §6, and your completed checklist. CHeKT provisions a test site for you, then arranges a joint test against a live site.
2. **Confirm with the dealer before go-live** that the target cameras' zone types are enabled for monitoring signals (§7.4), and whether CHeKT-side AI filtering is enabled on those cameras (§8.1).

Appendix A. Arming API

Most integrations never call the arming endpoints: the partner only needs to know that disarmed sites reject events (§8.1). Read this appendix if your product reads arming state for troubleshooting or controls arming.

A.1 Two arming models

The site list (§7.1) tells you which model a site uses:

- **Site system** (`is_partition_enabled: 0`) — one arming state for the whole site. Use the Arming endpoints.
- **Partition system** (`is_partition_enabled: 1`) — each partition has its own arming state, and a camera's alarm depends on the partition it belongs to. Use the Partition Arming endpoints, targeting `partition_ids`.

Implement both branches, and expect the partition branch to be the common one.

- All newly created sites default to the partition system.
- Enabling partitions is permanent. `is_partition_enabled` can change from 0 to 1 on an existing site, never back — re-read it when a binding starts failing.
- The partition system requires Bridge firmware v6.0.0 or higher and is enabled by the dealer in the Dealer Portal (Customers → Site Details → Partitions → Enable Partition).

A.2 The endpoint families are mutually exclusive

Branch on `is_partition_enabled` rather than trying one family and falling back to the other. Calling the wrong one is rejected by design:

Call	On a site-system site	On a partition-system site
GET <code>.../arming</code> , POST <code>.../arming/actions/*</code>	200	403 10301 — "This site uses a partition system..."
GET <code>.../partition-arming</code>	403 10303 — "This site does not use a partition system..."	200
POST <code>.../partition-arming/actions/*</code>	400 — no partitions exist	200
POST <code>.../arming/actions/*</code> with <code>partition_ids</code> in the body	403 — "partition_ids can not be specified because the site does not use a partition system"	200

Note the last row: the request *body* must vary with the site type, not just the endpoint.

A.3 Arming requests are all-or-nothing

A site's partitions can each be controlled by a different authority — CHeKT, Alarm.com, OpenEye, or an alarm panel voltage input — and some refuse API control. When an arming request covers a partition whose authority refuses API control, **the entire request is rejected and no partition changes**, including those that would have succeeded. This is a deliberate design decision: it keeps 200 meaning "everything requested was armed".

- **2xx** — every requested partition was armed.
- **4xx** — nothing changed.

The specification describes this as "best effort — depending on a partition's arming source, arming or disarming control may not be possible": the request can fail, not partially apply.

```
{
  "error": {
    "code": 13001,
    "message": "The arming request cannot be processed. See errors for details.",
    "detail": "10304 Partitions are controlled by the panel voltage input. partition_ids=686",
    "errors": [
      { "error_code": 10304, "message": "Partitions are controlled by the panel voltage input.",
        "partition_ids": [686] },
      { "error_code": 10305, "message": "Partitions are controlled by the ADC system.",
        "partition_ids": [669] }
    ]
  }
}
```

`error.errors[]` lists **reasons for rejection**, not per-partition outcomes. If you request `partition_ids: [366, 669]` and the array names only 669, partition 366 was not armed either — it is absent only because it had no rejection reason of its own.

Reason code	Meaning
10304	Partition is controlled by the panel voltage input
10305	Partition is controlled by the Alarm.com (ADC) system
10306	OpenEye partition arming setting is not valid
10423	Partition does not belong to this site
10404	Site not found

A.4 Arming action request body

Field	Type	Required	Description
user	string	yes	Who performed the action. Recorded in the activity log
partition_ids	integer[]	yes on the partition family	Non-empty array of positive integers. Optional on the site-level action of a partition site, where omitting it targets every partition

```
{ "user": "operator-01", "partition_ids": [12, 13] }
```

To control one partition deterministically, call the Partition Arming endpoints with that single partition.

Appendix B. Question Index

The questions partner engineering teams ask most often, and where each is answered.

#	Question	Where
1	Which integration path should we choose?	§2 — the cloud API
2	Our device is on the same network as the Bridge. Can we call it directly?	§2 — no; use the cloud API
3	Does our device need to support ONVIF?	§2 , Option 1
4	What has to be in place at the customer site?	§4
5	How does a dealer authorize us? Where does the API key come from?	§5
6	What do we have to provide to the dealer in our product?	§6
7	Which endpoint raises the alarm?	§7.4
8	How do we know which CHeKT camera to alarm?	§6.2 , §7.2
9	Who stores the mapping between our camera and CHeKT's?	§7.3 — you do
10	Can we record video without raising an alarm?	§7.5
11	Can we get the video back into our own UI?	§7.6
12	Does the alarm call return an event ID?	§8.3 — no; recover it via §7.6
13	Can we send a verification message to a contact?	§7.7
14	Which path prefix should we use — <code>/ext</code> , <code>/ref</code> or <code>/txid</code> ?	§7.8 — <code>/ext</code>
15	We got <code>200</code> , but the monitoring center saw nothing. Why?	§8.1
16	Our event was rejected even though the key is valid. Why?	§8.1 , §9
17	We raised an alarm by zone and it appeared on a different camera. Why?	§8.2
18	We fired several alarms at once — will they all produce clips?	§8.6
19	What do the error codes mean, and what should we retry?	§9
20	Are there rate limits?	§9.4
21	How do partitions change the behavior? Can we arm or disarm through the API?	Appendix A
22	How do we test before going live, and how do we get certified?	§11