

StreamGate Public API Integration Guide

StreamGate 1.9.7 · Customer delivery document

This guide describes the REST API used by third-party systems to integrate with StreamGate. It covers licensing, authentication, scopes, resources, examples, error handling, security, and version compatibility.

Item	Value
Applicable StreamGate versions	1.8.2, 1.9.1, 1.9.2, 1.9.3, 1.9.4, 1.9.6, 1.9.7
API version	v1
Document version	1.2
Updated	2026-10-13
Base URL example	http://streamgate.example:18080
API prefix	/api/public/v1
Media type	application/json; charset=utf-8
Authentication	Authorization: Bearer <API_TOKEN>
Time format	ISO 8601; server timestamps use UTC

[streamgate.example](#) is a placeholder. Replace it with the deployed StreamGate host name or IP address and retain the configured Web port.

1. Scope

The Public API is available to Professional and Enterprise deployments. It exposes devices, channels, runtime state, playback URLs, recording indexes, alarms, and management endpoints for devices, channels, recording, GB28181, WebHook, and MQTT.

Integration	Direction and purpose	Authentication
REST API	The client requests the gateway to read or modify resources	Bearer token and required scopes
WebHook	The gateway POSTs events to the client's HTTP receiver	Receiver verifies the HMAC signature
MQTT	The gateway publishes to a broker/topic; the client subscribes	Broker credentials, topic permissions and configured QoS

These are distinct interfaces. API routes [/integrations/webhooks](#) and [/integrations/mqtt](#) manage subscriptions and delivery settings; they are not event receiver addresses. An API token does not replace a WebHook signing secret or MQTT credentials.

Professional and Enterprise packages include [streamgate-public-api-v1.openapi.yaml](#), the authority for request and response schemas. This guide explains integration behavior.

2. Version compatibility

Capability	1.8.2	1.9.1	Required scope
List channels	Yes	Yes	channels:read
Query recording index	Yes	Yes	recording:read
Query alarms	Yes	Yes	alarms:read
Acknowledge or clear alarms	Yes	Yes	alarms:write
List device identities	No	Yes	devices:read
Query channel runtime state	No	Yes	status:read
Get playback URLs	No	Yes	playback:read
Manage devices and ONVIF	No	Professional	devices:write
Configure channels and PTZ	No	Professional	channels:write , ptz:write
Manage recording tasks and policy	No	Professional	recording:read , recording:write
Manage GB28181	No	Professional	gb28181:read , gb28181:write
Manage WebHook, MQTT, and delivery logs	No	Professional	integrations:read , integrations:write

Clients must ignore unknown response fields so compatible releases can add optional information. For a mixed 1.8.2/1.9.1/1.9.2/1.9.3/1.9.4 fleet, probe the required endpoint and downgrade behavior when a 1.8.2 instance returns [404](#).

3. License, token, and scopes

In current 1.9.7, Public REST API, WebHook, and MQTT are unavailable in Free and Standard. A valid Professional trial, Professional, or Enterprise software license enables them, subject to the [openapi](#) entitlement and runtime package. Evaluation video has no watermark; trials are only for permitted non-production evaluation, with term, capacity, and feature scope defined by the issued license. REST calls and REST integration management also require a valid API token with the corresponding scopes; WebHook signature verification, MQTT credentials, and topic permissions are separate requirements. Analysis licensing does not replace software entitlements. Internal console endpoints are not Public API endpoints. Historical 1.8.2 endpoints still require an OpenAPI-enabled Professional or Enterprise license; version support does not grant access rights.

Create a token in [System Settings > Open API > API Tokens](#) while signed in as an administrator. Name the client system, select only the required scopes, and store the complete token immediately. The complete token is displayed once; StreamGate retains and later displays only its prefix. If it is lost, revoke it and create a replacement.

Scope	Purpose
channels:read	Read channel inventory and channel configuration.
recording:read	Read recording index, tasks, and policy.
alarms:read	Read alarm state.

Scope	Purpose
alarms:write	Acknowledge or clear alarms.
devices:read	Read device identity and redacted device configuration.
status:read	Read observed channel distribution state.
playback:read	Read channel playback URLs.
playback:warm	Keep a channel source ready briefly; separate from URL read permission.
devices:write	Create, update, delete, batch-operate, and discover devices.
channels:write	Create, update, delete, batch-operate, and change channel demand mode.
ptz:write	Execute PTZ commands; intentionally separate from channel configuration.
recording:write	Manage recording tasks and policies.
gb28181:read	Read GB28181 platforms, exposure bindings, and upstreams.
gb28181:write	Manage GB28181 platforms, exposure bindings, and upstreams.
integrations:read	Read WebHook, MQTT, event-type, and delivery-log state.
integrations:write	Manage and test WebHook/MQTT and clear delivery logs.
exports:read	Read redacted StreamGate runtime events.
*	All Public API permissions; reserve for controlled trusted integrations.

Send the token in the request header:

```
Authorization: Bearer <API_TOKEN>
Accept: application/json
Content-Type: application/json
```

GET requests may omit **Content-Type**. Never put the token in a URL, query string, browser script, screenshot, or ordinary application log.

4. Quick verification

```
curl --request GET \
  "http://streamgate.example:18080/api/public/v1/channels" \
  --header "Authorization: Bearer <API_TOKEN>" \
  --header "Accept: application/json"

$baseUrl = "http://streamgate.example:18080"
$headers = @{ Authorization = "Bearer <API_TOKEN>" }

Invoke-RestMethod `
  -Uri "$baseUrl/api/public/v1/channels" `
  -Headers $headers `
  -Method Get
```

The expected status is **200** and the body contains a **channels** array. An empty array means no channel is configured; it is not an API failure.

5. Baseline read resources

Channels

GET </api/public/v1/channels> requires `channels:read`. It returns configured channels without device credentials or source URLs.

Field	Type	Meaning
<code>channels[].id</code>	string	Stable logical channel identifier.
<code>channels[].endpointId</code>	string	Related ingest endpoint, when present.
<code>channels[].deviceId</code>	string	Related device, when present.
<code>channels[].name</code>	string	Display name.
<code>channels[].enabled</code>	boolean	Whether the channel is enabled.
<code>channels[].inputType</code>	string	Input protocol type.
<code>channels[].outputStreamId</code>	string	Distribution stream identifier.

Recordings

GET </api/public/v1/recordings> requires `recording:read`. Supported query parameters are defined in OpenAPI and include channel, time range, pagination, and sort order. Treat all timestamps as UTC ISO 8601 values. A recording path is server-side deployment information and may be sensitive.

Alarms

GET </api/public/v1/alarms> requires `alarms:read`. Filters include status, channel, severity, time range, and pagination. Use the returned `dedupeKey` for state-changing operations:

- POST </api/public/v1/alarms/{dedupeKey}/ack> with `alarms:write` acknowledges an alarm.
- POST </api/public/v1/alarms/{dedupeKey}/clear> with `alarms:write` clears an alarm.

After a timeout on either write operation, query the alarm again before retrying.

Devices

GET </api/public/v1/devices> requires `devices:read` and StreamGate 1.9.1 or later. It returns identity fields such as `id`, `name`, `vendor`, `protocol`, and `enabled`; credentials and private source URLs are omitted.

Runtime status

GET </api/public/v1/status> requires `status:read`. It returns the distribution state observed by StreamGate. It does not actively probe the source device and is not equivalent to device online state.

Known values are `unknown`, `starting`, `active`, `draining`, `idle`, and `error`. Clients must allow compatible releases to add values.

6. Playback URLs

GET /api/public/v1/channels/{id}/playback-urls requires `playback:read`. Optional query parameter `quality` accepts `auto`, `high`, or `fluent`. Omission uses the stored `viewing.defaultQuality`; an explicit value overrides it. Older channels without viewing configuration retain their single-stream behavior. With `auto`, clients may send `viewportWidth` and `viewportHeight` from 1 through 8192. A supplied width of at most 640 or height of at most 360 selects an available substream; larger windows or no viewport signal prefer the main stream. An unavailable substream falls back to the main stream.

In System Settings > Media & Playback, Console playback stream sets `media.defaultViewingQuality`: `auto/high/fluent` mean Auto, Main stream, and Substream. This applies to console preview, details, and Standard/Professional video walls. Auto selects by window size and presentation; an unavailable substream falls back to main. The channel UI no longer has a default-playback selector. Legacy `viewing.defaultQuality` remains only for Public API playback-URLs requests without `quality`; explicit `quality=high/fluent/auto` still overrides it. The system console choice does not change Public API behavior, recording sources, or GB28181 delivery.

Calling this endpoint does not start source ingest and does not issue playback credentials.

Field	Meaning
<code>id</code> , <code>enabled</code> , <code>outputStreamId</code>	Logical channel identity and current configuration.
<code>httpFlv</code>	HTTP-FLV URL, empty when the protocol is disabled.
<code>websocketFlv</code>	WebSocket-FLV URL, empty when disabled.
<code>hls</code>	HLS URL, empty when disabled.
<code>webRtc</code>	WebRTC playback-page URL, empty when disabled.
<code>whep</code>	WHEP endpoint URL, empty when disabled.
<code>requestedQuality</code> , <code>actualQuality</code>	Requested and selected quality tiers.
<code>qualitySource</code> , <code>qualityReason</code> , <code>qualityFallback</code>	Selection source and explanation.
<code>fluentAvailable</code>	Whether a substream output is available.
<code>playbackTokenRequired</code>	Whether separate playback authorization is still required.

An API token authenticates the Public API only. It does not replace a Web Console session or playback credential. When `playbackTokenRequired` is true, obtain a valid credential through the deployment's configured playback-authentication flow.

A multi-window client should request `?quality=fluent` for grid cells using the substream, stop the existing player and request `?quality=high` for the main stream when a cell is enlarged, then switch back to `fluent` when it returns to the grid. Always use `actualQuality`; if no substream is configured, the server falls back to the main stream and reports the reason in `qualityFallback`.

`high` means the channel main stream and `fluent` means the substream. The existing `quality` parameter, `high/fluent` values, response fields, and playback URL keys remain compatible. Selecting a stream does not create a transcode policy. The channel main stream uses the selected input, which may itself be a camera substream; it is not necessarily the camera's physical main stream. A substream can use another device stream or independent encoding of the same original input, not re-encoding of the main output. Independent policies and legacy behavior are documented in Dual-stream output configuration below.

To play H.265 in an incompatible browser, enable compatibility transcoding for the selected output; changing [quality](#) alone does not ensure H.264.

Optional native HLS compatibility: since StreamGate 1.9.6, only clients that cannot use MSE and require native HLS append [hls_compat=fmp4](#) to the returned Gateway [hls](#) playback URL. Use [?](#) when no query exists, otherwise [&](#); preserve every original query parameter, especially playback credentials. MSE / hls.js players do not append it. This selects complete fMP4 fragments for compatibility, not an LL-HLS low-latency promise or a substitute for codec compatibility transcoding. It adds no REST resource and does not change playback credentials or their signature contract. Do not add it to the [playback-urls](#) API request.

Available since StreamGate 1.9.6. Optional: call [POST /api/public/v1/channels/{id}/warm](#) only when a page can prepare an on-demand channel before the user opens its player. Normal playback needs no warm call; always-on channels already keep their source connected. Use a token with the separate [playback:warm](#) scope and send the same [quality](#) value as the subsequent playback request. [{"ttlMillis":15000}](#) requests 15 seconds, capped at 30 seconds; the response includes [expiresAt](#). Start playback after a successful response. If warming fails, the client can still use the normal playback flow. Do not warm every channel or keep refreshing it in a background loop. Warming keeps the source ready but does not pre-generate LL-HLS segments or replace playback credentials. Versions before 1.9.6 return 404 for this route.

Player recovery and source changes

StreamGate 1.9.6 adds optional [channel.media_changed](#) notifications through the existing WebHook/MQTT integration and optional [channels\[\].media](#) in [GET /api/public/v1/status](#). Earlier versions may omit [media](#); clients must continue to accept the existing [id](#), [enabled](#), and [state](#) response. There is no new SSE server or browser subscription endpoint. Notifications supplement, rather than replace, player events and playback-progress monitoring:

- WebRTC: [connectionstatechange](#) to [failed](#), or a media track's [ended](#) event. Allow time for transient [disconnected](#) states to recover.
- hls.js: recover or recreate the player for [Hls.Events.ERROR](#) with [fatal=true](#). Let the library recover non-fatal errors.
- HTTP-FLV/WS-FLV: handle the playback library's error and live-connection completion events; dispose of the old instance before reconnecting.

Also detect sustained lack of playback progress, excluding paused/hidden players. Keep one retry task with backoff and stop it when the user closes the player. A [video.resize](#) event needs only a layout adjustment if playback is still progressing. Reuse stable URLs for the same channel; request new URLs/credentials after quality changes or expiry. Reconnecting cannot make an unsupported H.265 source compatible. Optional [warm](#) is neither a recovery notification nor a credential refresh.

Optional media-change notifications (1.9.6)

Configure the existing WebHook or MQTT endpoint with `eventTypes:["channel.media_changed"]` to subscribe only to media-change notifications. StreamGate 1.9.6 lists this exact type in [GET /api/public/v1/integrations/event-types](#). Existing event types remain accepted and may be combined with it; `eventTypes:["*"]` still delivers all events, while `channel.*` is not supported. Omitting/emptying `eventTypes` keeps the existing default list and does not automatically add media-change notifications. Public integration configuration keeps `integrations:read/integrations:write`; status keeps `status:read`. Valid Professional/Enterprise software licensing with `openapi` remains required, and media delivery checks that entitlement again. No new license feature or playback permission is granted.

State field (<code>channels[].media[]</code>)	Meaning
<code>outputStreamId, quality</code>	Output identity and high/fluently tier; compare only the same channel/output.
<code>current, observedAt</code>	Observation validity and UTC collection time. Failed/expired Core observations retain old values with <code>current=false</code> ; they do not prove a stop.
<code>available</code>	An active video descriptor was observed, not proof of device health, decoded frames, or every protocol being ready.
<code>mediaGeneration</code>	Opaque Gateway comparison marker. Do not parse or order it; re-establish the baseline after restart.
<code>format</code>	videoCodec , width , height , and origin (source or applied). Omitted when unavailable. Not a per-protocol wire measurement; audio adaptation is not covered.
<code>effectiveVideo</code>	Optional adopted encoder creation values: codec , width , height , fps , bitrateKbps , keyframeIntervalMs . Not instantaneous traffic or a quality guarantee.
<code>publisherGenerations, outputGeneration, selection</code>	Optional Core-local publisher/output generations as decimal strings and selected output policy. Their counters may reset; use mediaGeneration for comparison.

The observer reads one aggregate Core state in the existing serial media loop (nominal two-second interval, not a latency SLA); it never starts a source or probes each channel. Public status reads only this cache. Unobserved/old-Core media may be absent; observations older than ten seconds are not current. State is bounded to at most two outputs per configured channel; deleted channels are removed without a final notification.

Deduplication ignores traffic rates, packet/frame/queue counters, and requested-but-not-adopted settings. Codec/dimensions, adopted encoder settings/instance, source restart identity, and publisher/output generation changes can produce a new marker. Main-stream and substream outputs have separate keys; a substream-only output change does not change the main-stream baseline. An observed stop/resume is reported, including same-format restart. First observation uses `reason=observed`; later reasons are `changed`, `stopped`, or `resumed`, with future values allowed. Initializing/stale encoder data does not invent a new format.

Current Core `videoEncoding.instanceId` and `encoderGeneration` participate in the internal comparison. A new adopted bitrate caused by a source estimate can therefore notify even when codec/dimensions are unchanged. Changes seen in one aggregate snapshot produce one event per output; changes exposed in separate observations may produce separate events.

The existing envelope retains `eventId`, `type`, `occurredAt`, `deploymentId`, `nodeId`, `mediaResourceId`, `source`, `action`, `severity`, `message`, and string-valued `payload`. Example:

```
{
  "eventId": "evt-example",
  "type": "channel.media_changed",
  "occurredAt": "2026-10-01T08:00:00Z",
  "mediaResourceId": "ch-001",
  "source": "channel",
  "action": "media_changed",
  "payload": {
    "outputStreamId": "ch-001_fluent",
    "quality": "fluent",
    "mediaGeneration": "media-example-2",
    "previousMediaGeneration": "media-example-1",
    "available": "true",
    "reason": "changed",
    "observedAt": "2026-10-01T08:00:00Z",
    "videoCodec": "h264",
    "width": "640",
    "height": "360",
    "formatOrigin": "applied",
    "bitrateKbps": "400",
    "publisherGenerations": "\\hls-rtsp\\\"2\\\""
  }
}
```

WebHook keeps existing HMAC headers/retries/delivery logs; MQTT keeps its existing topic and QoS behavior. These are best-effort sampled notifications, not a durable change log: short changes between observations, queue overflow, receiver failure, or process restart can omit notifications; retries can duplicate them. Deduplicate by `eventId` and compare the channel/output marker. Optional low-frequency status polling can reconcile the latest current state, but cannot recover missing intermediate changes. A stopped notification is not an instruction to call `warm` or restart the source. On changed/resumed media, refresh player metadata and use one bounded player recovery task only when playback needs it; do not reconnect a progressing player solely for a bitrate-setting or resize change.

A new publisher generation may end an existing connection (for example, RTSP EOF) and require reconnection. This does not certify clean decoding: do not dismiss non-EOF decoder corruption as an expected generation transition.

7. Device and channel management

These endpoints require StreamGate 1.9.1 or later, Professional or Enterprise. Write requests may submit device credentials. Responses remove passwords, password references, keys, URL user information, and sensitive query parameters.

Method and path	Scope	Purpose
POST /devices	devices:write	Create a device.
GET /devices/{id}	devices:read	Read redacted device configuration.
PUT /devices/{id}	devices:write	Update a device.
DELETE /devices/{id}	devices:write	Delete a device.
POST /devices/batch	devices:write	Batch enable, disable, or delete.
POST /devices/onvif/discover	devices:write	Discover ONVIF devices.
POST /devices/onvif/resolve-stream	devices:write	Resolve an ONVIF media stream.
POST /channels	channels:write	Create a channel.
GET /channels/{id}	channels:read	Read channel configuration.
PUT /channels/{id}	channels:write	Update a channel.
DELETE /channels/{id}	channels:write	Delete a channel.
POST /channels/batch	channels:write	Batch enable, disable, ondemand, always_on, or delete.
POST /channels/{id}/ptz	ptz:write	Execute a PTZ command.

Dual-stream output configuration

Channel creation and updates require `channels:write`. Root `output` controls the main stream; optional `viewing.fluent.output` controls the substream independently. Neither changes camera encoding. Missing fields in older configurations retain the behavior in the compatibility table below.

Field	Meaning
<code>viewing.defaultQuality</code>	Legacy default only for Public API playback-URLs requests without <code>quality</code> , not console preview/details or walls. <code>auto/high/fluent</code> remain; high selects main, fluent selects substream. Stored auto/fluent requires a configured substream. Explicit query quality overrides it; the channel UI no longer exposes this selector.
<code>viewing.fluent.source</code>	<code>profile</code> : Device substream Profile with <code>profileToken</code> ; <code>url</code> : Device substream URL with <code>sourceUrl</code> ; <code>output</code> : transcode from the original main input, without either source-detail field.
<code>output.mode</code> , <code>viewing.fluent.output.mode</code>	Empty/omitted: Use system; <code>source</code> : Source stream; <code>compatible</code> : Compatibility; <code>custom</code> : Always transcode. The main stream and device substreams allow all four. Generated substreams allow only compatible/custom. The display name does not change the custom value.
Each output's <code>video</code>	<code>compatible/custom</code> accept <code>bitrateKbps</code> , <code>qualityPreset</code> , <code>automaticPolicy</code> , <code>automaticBitrate</code> , <code>hevcBitrateReservePercent</code> , <code>maxWidth/maxHeight</code> , <code>maxFps</code> , <code>keyframeIntervalMs</code> (milliseconds), and <code>encoderPreference</code> (auto/hardware/software). Compatibility converts only H.265 to H.264; these parameters are unused for H.264 passthrough. Do not send video overrides for source/Use system.
<code>video.automaticPolicy</code>	Optional <code>economy/balanced/quality</code> objects with <code>sourcePercent</code> , <code>referenceBitrateKbps</code> , and <code>referenceCeilingKbps</code> ; nonzero policies take priority over legacy strength. Defaults, validation, and whole-object inheritance are described below.
<code>video.hevcBitrateReservePercent</code>	Optional integer 0..200, applied only to automatic bitrate for H.265-to-H.264 transcoding. Omission inherits the system; explicit 0 overrides it with no reserve. Independent of <code>automaticPolicy</code> ; system omission defaults to 35.
<code>video.automaticBitrate</code>	Legacy optional object with <code>economyPercent</code> , <code>balancedPercent</code> , and <code>qualityPercent</code> ; each is an integer 0 or 50..150. Percentages adjust built-in automatic targets, not fixed kbps or image-quality scores. See whole-object inheritance below.

Transcode from main input shares the original input and encodes independently, not in series after the main-stream encoder. The editor defaults to custom, preserving source dimensions/frame rate and inheriting the system preset. The following channel request fragment explicitly selects 640 x 360, 15 fps, and economy; preserve other required fields when updating. Dimensions are aspect-ratio-preserving reduction bounds and frame rate is a ceiling, without upscaling or frame interpolation. Changing the substream policy does not overwrite the main-stream policy.

Dual-stream example

```
{
  "output": {
    "mode": "compatible", "video": { "qualityPreset": "balanced", "hevcBitrateReservePercent": 0 },
    "viewing": {
      "defaultQuality": "auto",
      "fluent": {
        "source": "output",
        "output": {
          "mode": "custom",
          "video": { "qualityPreset": "economy", "maxWidth": 640, "maxHeight": 360, "maxFps": 15 }
        }
      }
    }
  }
}
```

Here the main stream explicitly disables the H.265 bitrate reserve; the substream omits the field and inherits it from the system. This does not change their output modes or playback selection.

Configuration	Main-stream policy	Substream policy
profile/url, fluent.output absent	Root output	Root output
profile/url, explicit fluent.output={}	Root output	Independently use system
output source, fluent.output absent	Use system	Root output (legacy special case)
output source, explicit fluent.output	Root output	fluent.output, compatible/custom required

An absent field is not an empty object. Reads do not migrate old configurations. The first editor save converts explicitly while preserving both streams' previous behavior: device substreams receive a copy of the old root policy; generated substreams receive the old root policy while the main-stream policy is saved as Use system. Existing client fields and playback URLs remain accepted.

Automatic bitrate and on-demand ingest

Both outputs' transcoding `video.qualityPreset` accepts "", "economy" (Save bandwidth), "balanced" (Balanced), or "quality" (Prefer quality). Omission or empty follows system `media.videoQualityPreset` (default Balanced); explicit presets override it. A positive manual `video.bitrateKbps` takes priority and ignores automatic policy and H.265 reserve.

In System Settings > Media & Playback, choose Save bandwidth, Balanced or Prefer quality. The adjacent gear opens Transcode settings for each preset's quality strength and optional bitrate limit, plus the shared keyframe interval. Select the global encoder backend under Advanced playback & resources below.

Automatic presets preserve source resolution and frame rate. Set dimensions or a frame-rate limit explicitly in the channel's main/substream output when needed. Main and substream Use system independently inherit the global transcoding mode and preset; enabling transcoding does not create an unconfigured substream.

Preset	Default quality strength	x264 CRF	Intel QSV QVBR quality
Save bandwidth <code>economy</code>	4	28	28
Balanced <code>balanced</code>	6	25	25

Preset	Default quality strength	x264 CRF	Intel QSV QVBR quality
Prefer quality quality	8	22	22

Higher quality strength preserves more detail and usually needs more bitrate. Strength is not a native encoder value: x264 uses CRF+VBV, QSV uses QVBR when supported, and NVENC/AMF use their supported quality-priority VBR modes. OpenH264 and other backends without the corresponding mode fall back to a bitrate budget, not equivalent CRF quality. Backend values do not imply equal quality or recovery of detail already lost at the source.

The three [economy/balanced/quality](#) presets are stored in [media.videoTranscodeParameters.automaticPolicy](#). Each [qualityLevel](#) is an integer 1..10; zero/omission uses the built-in value, with nondecreasing preset order. Optional [maxBitrateKbps](#) accepts zero or an integer 256..64000 kbps. Zero/empty lets the system estimate a budget. A limit is neither a fixed bitrate nor a hard network cap including audio, packaging and retransmission. Unsupported quality modes are reported as [bitrate-fallback](#) in actual output.

An omitted, empty or all-zero channel [video.automaticPolicy](#) inherits the whole system policy; a nonzero object replaces it, using built-in defaults for missing fields. A positive channel [bitrateKbps](#) selects manual bitrate and bypasses automatic presets. All transcoding outputs use the global encoder backend; legacy channel [encoderPreference](#) remains readable but does not override it. A zero channel keyframe interval inherits the global value, whose default is 1000 ms.

Compatible mode passes an H.264 device stream through and transcodes H.265 to H.264. Transcode from main input explicitly creates an independently encoded output even when the main source is H.264. These source types differ; the generated substream does not re-encode the main output's encoded data.

Legacy [sourcePercent/referenceBitrateKbps/referenceCeilingKbps](#), [automaticBitrate](#) and [hevcBitrateReservePercent](#) remain accepted for internal budgeting/fallback, not quality strength. Budgets consider measured source bitrate and actual output dimensions/frame rate; before measurement they use a format estimate. Legacy [maxWidth/maxHeight/maxFps](#) inside automatic presets no longer change the picture. Explicit channel [video](#) geometry and frame rate remain effective.

Cancel leaves the page unchanged; Apply edits the draft, and Save system settings makes it effective. Applying without edits creates no unsaved state. Shared-encoding edits preserve a legacy fixed bitrate; editing the automatic policy or choosing Use automatic bitrate switches to automatic mode. Read runtime state for the adopted backend, rate-control mode and native quality value. No Public API system-settings endpoint is added.

Legacy [automaticBitrate](#) remains compatible and is used only without an effective nonzero new policy after inheritance. Zero/omission means built-in 100%; explicit 100 remains an override. Only an absent, empty, or all-zero object inherits the entire system strength object. Nonzero objects replace it as a whole, with omitted/zero components using built-in 100%, not individual system fields. For example, `{"balancedPercent":100}` overrides a legacy system value of 90 or 120 and uses built-in 100% for the other presets. Adjusted targets must retain Economy, Balanced, Quality ordering; reversed combinations are rejected. This is neither fixed kbps nor an image-quality score and is no longer the current system gear UI.

Compatibility uses parameters only for H.265 transcoding. H.264 passthrough and source mode do not use automatic policy and are not transcoded because of a preset. Saving a global preset or policy may briefly reconnect active outputs that use the changed value; explicit channel policies and manual bitrates are not overwritten, and unused preset changes do not rebuild the current output. Parameters guarantee neither network bandwidth nor original image quality, add no scope or license feature, and a successful save is not proof of encoder adoption; read current media state when available.

Create-device example

```
POST /api/public/v1/devices
Authorization: Bearer <API_TOKEN>
Content-Type: application/json

{
  "name": "East Gate Camera",
  "vendor": "generic",
  "protocol": "onvif",
  "address": "192.0.2.20",
  "username": "operator",
  "passwordRef": "<DEVICE_PASSWORD>",
  "enabled": true
}
```

`onDemand=true` or batch `{"action":"ondemand","ids":["ch-001"]}` means On-demand ingest enabled: pull while preview, distribution, recording, analysis, or GB28181 needs the source, and allow stopping after idle. `onDemand=false` / `always_on` means On-demand ingest disabled: keep the input connected while the channel is enabled. API fields and values are unchanged; active tasks can still keep an on-demand source connected.

8. Recording management

Method and path	Scope	Purpose
GET /recording/tasks	recording:read	List recording tasks.
POST /recording/tasks	recording:write	Create a recording task.
GET /recording/tasks/{id}	recording:read	Read one recording task.
PUT /recording/tasks/{id}	recording:write	Update a recording task.
DELETE /recording/tasks/{id}	recording:write	Delete a recording task.
GET /recording/policy	recording:read	Read the default recording policy.
PUT /recording/policy	recording:write	Update the default policy.
POST /recording/policy/apply	recording:write	Apply the policy to selected or all channels.

In 1.9.6, select the recording target in two steps: the main stream or substream, then the original or processed data for that stream.

Field	Values	Selection
streamRole	default / secondary	Main stream / substream
mediaSource	source / output	Original / processed data for the selected stream

An independent device substream supports both data selections. A substream generated by transcoding the main input supports only [secondary/output](#). Selecting [secondary](#) without a configured substream, or another invalid combination, returns [400](#); it does not silently fall back to the main stream. Changing the target does not change recording enablement, schedules or retention. Existing segments keep their original source information.

For older tasks, an omitted [streamRole](#) retains the main stream. An omitted or empty [mediaSource](#) retains the existing processing policy. Preserve unmodified fields returned by the read operation when updating an older task.

Recording directories are resolved and validated on the StreamGate server, not on the calling computer. When a recording root is configured, a task directory must remain within that root.

9. GB28181 management

Each resource group supports collection [GET/POST](#) and item [GET/PUT/DELETE](#). Read operations require [gb28181:read](#); writes require [gb28181:write](#).

Resource	Purpose
/gb28181/platforms	Configure local GB28181 service platforms.
/gb28181/exposure-bindings	Publish StreamGate channels into a GB28181 catalog.
/gb28181/upstreams	Register and cascade to upstream platforms.

Use the OpenAPI contract for complete request schemas. Platform or upstream passwords may be submitted on writes; [passwordRef](#) is not returned in responses.

In 1.9.6, choose one upstream media policy for all shared channels. There are no per-channel overrides.

Media policy	Fields	Upstream requests
Single stream: main	mediaStreamMode=single , singleStreamRole=default	Supply the channel's main output
Single stream: substream	mediaStreamMode=single , singleStreamRole=secondary	Supply the configured substream output
Main and substream	mediaStreamMode=dual	No stream number or number 0 selects main; number 1 selects substream

In single-stream mode, valid main/substream requests receive the selected stream. A channel without a configured substream supplies main under these policies; no substream is created implicitly. Other stream numbers are rejected. Older configurations with both fields unset retain their existing behavior. This policy applies to live upstream media, not recording targets or transcoding parameters.

10. WebHook, MQTT, and runtime events

The following REST API routes manage delivery. Events are sent to the configured WebHook receiver or MQTT broker/topic, using the direction and authentication described at the start of this guide.

Method and path	Scope	Purpose
GET /integrations/event-types	integrations:read	List subscribable event types.
GET /integrations/webhooks	integrations:read	List WebHooks.
POST /integrations/webhooks	integrations:write	Create or update a WebHook.
DELETE /integrations/webhooks/{id}	integrations:write	Delete a WebHook.
POST /integrations/webhooks/{id}/test	integrations:write	Send a real test event.
GET /integrations/mqtt	integrations:read	Read MQTT configuration.
PUT /integrations/mqtt	integrations:write	Update MQTT configuration.
POST /integrations/mqtt/test	integrations:write	Send a real test message.
GET /integrations/deliveries	integrations:read	Query delivery logs by endpoint, type, and status.
DELETE /integrations/deliveries	integrations:write	Clear delivery logs.
GET /events	exports:read	Query redacted runtime events.

```
{
  "name": "Operations Alarm Receiver",
  "enabled": true,
  "url": "https://ops.example.com/streamgate/events",
  "eventTypes": ["alarm.*", "recording.error", "channel.stream_error"],
  "hmacSecret": "<SHARED_SIGNING_SECRET>",
  "timeoutSeconds": 5,
  "maxAttempts": 3
}
```

`hmacSecret` and the MQTT `password` are write-only. Query and save responses do not return plaintext values. Test endpoints perform real network delivery; use a dedicated receiver during integration.

11. Error handling

Error responses are JSON. Branch on the HTTP status and stable structured fields, not on the complete human-readable error text.

```
{
  "error": "api token scope denied",
  "scope": "alarms:write"
}
```

Status	Meaning	Recommended action
400	Invalid parameter or JSON body.	Correct types, ranges, and JSON; do not retry unchanged.
401	Missing, invalid, disabled, or revoked token.	Check the Authorization header or issue a replacement token.
403	License entitlement or token scope is insufficient.	Check the license page and token scopes.
404	Endpoint or resource does not exist.	Check version, resource ID, and URL encoding.
405	HTTP method is not allowed.	Check GET, POST, PUT, and DELETE usage.

Status	Meaning	Recommended action
500	Server-side processing failed.	Record request time and response summary, then inspect StreamGate logs.

GET requests may retry transient network errors and [5xx](#) with bounded exponential backoff, for example 1, 2, and 4 seconds. Do not unconditionally retry [400](#), [401](#), [403](#), or [404](#). After a timeout on a state-changing request, query current state before deciding to retry.

12. Security and deployment

- Access StreamGate through a trusted network. For cross-network access, use a controlled HTTPS reverse proxy or secure tunnel.
- Create a separate token for each client system and grant only required scopes.
- Store tokens only in server-side secret configuration. Do not commit them to source control.
- Log only the token prefix, never the full Authorization header.
- Treat recording paths, alarm extensions, device identity, and playback URLs as deployment-sensitive data.
- Keep management and media access controls separate. A Public API token does not grant playback when separate playback authorization is enabled.
- Revoke unused tokens and review delivery destinations periodically.

13. Integration checklist

- 1 The client server can reach the StreamGate Web host and port.
- 2 The license page shows Professional, Enterprise, or the required valid API entitlement.
- 3 The complete token is stored in the client's protected configuration.
- 4 The token has the minimum scopes required by the selected endpoints.
- 5 [GET /channels](#) returns 200.
- 6 Endpoints introduced in 1.9.1 are not assumed on a 1.8.2 instance.
- 7 The client handles timeout, [401](#), [403](#), [404](#), and [5xx](#) explicitly.
- 8 Logs, screenshots, and exports contain no complete token or device password.
- 9 WebHook and MQTT test deliveries use a dedicated receiver and are reviewed before production enablement.
- 10 Playback is tested with the deployment's actual playback-authentication policy.

14. Companion files

[GET /api/public/v1/openapi.json](#) returns the machine-readable contract when the running instance has a valid Professional or Enterprise license. Use it for version probing and tool import. Professional and Enterprise packages also include [docs/streamgate-public-api-v1.openapi.yaml](#), which can be imported into Swagger Editor, Postman, or a compatible code generator.

The Postman collection is delivered only when specifically requested for integration work. The HTML source is a documentation-maintenance asset and is not included in the runtime package.

All host names, resource identifiers, tokens, credentials, and business data shown here are examples. Replace them with values from the actual deployment.